ALGEBRA W SAGEMATH

Część 1: LICZBY ZESPOLONE

WITALI BUŁATOW

Politechnika Białostocka

WITALI BUŁATOW

ALGEBRA W SAGEMATH
CZĘŚĆ 1: LICZBY ZESPOLONE

Politechnika Białostocka
OFICYNA WYDAWNICZA POLITECHNIKI BIAŁOSTOCKIEJ
BIAŁYSTOK 2026

Recenzent:
dr Anna Łyczkowska-Hańćkowiak

Redaktor naukowy dyscypliny matematyka:
prof. dr hab. inż. Zbigniew Bartosiewicz

Redakcja i korekta językowa:
Katarzyna Duniewska

Skład, grafika i okładka:
Marcin Dominów

ISBN 978-83-68673-30-2 (e-Book)
DOI: 10.24427/978-83-68673-30-2

Publikacja jest dostępna w Internecie na stronie Oficyny Wydawniczej PB.
Oficyna Wydawnicza Politechniki Białostockiej
ul. Wiejska 45C, 15-351 Białystok
www.pb.edu.pl

Spis treści

1. Wstęp

Materiały zawarte w tej książce powstały podczas prowadzenia zajęć w ramach pracowni specjalistycznej z analizy i algebry na Politechnice Białostockiej. Wszystko zaczęło się, gdy na uczelni poszukiwano osoby do poprowadzenia zajęć z matematyki ze znajomością języka **Python**. Gdy podjąłem się tego zadania, okazało się, że wcześniejsze zajęcia realizowano w systemie **SageMath**, o którego istnieniu nawet nie miałem pojęcia. W czasie moich studiów matematyka ograniczała się do tradycyjnych wykładów przy tablicy, bez wizualizacji omawianych pojęć. Miałem jednak dość dobrą znajomość **Pythona** – kilka miesięcy wcześniej poświęciłem wiele godzin na jego naukę ze źródeł dostępnych w internecie.

Początkowo na zajęciach używałem bibliotek **NumPy** i **Matplotlib** z języka **Python** do wizualizacji pojęć z analizy matematycznej. Po upływie miesiąca zapoznałem się z możliwościami systemu **SageMath**, aby pokazać bardziej skomplikowane zagadnienia. Okazało się, że za pomocą języka **Sage** można wykonywać te same rysunki łatwiej i przy użyciu mniejszej ilości kodu. Ponadto oferuje on dodatkowe możliwości, takie jak interaktywne wizualizacje 3D czy obliczenia symboliczne. **SageMath** jest również naturalnym rozszerzeniem **Pythona**, który jest intuicyjnym językiem programowania, pozwalającym na stosowanie wielu kreatywnych rozwiązań.

W ten sposób, w miarę prowadzenia zajęć, zaczęło powstawać wiele materiałów pomocniczych do nauki matematyki wyższej. Na początku było to trudne – przedmiot został niedawno wprowadzony na uczelni, a w internecie brakowało gotowych rozwiązań. To spowodowało, że kod oraz układ materiału zacząłem tworzyć samodzielnie, rozwiązując problemy dotyczące zagadnień, które chciałbym zrealizować. Ten tryb pracy bardzo mi odpowiadał, ponieważ pozwalał na nieustanne wprowadzanie nowych pomysłów. Po pewnym czasie okazało się, że zajęcia cieszą się sympatią studentów i pomagają im w nauce – to sprawiało mi największą radość i motywowało do dalszego udoskonalania materiałów.

Te doświadczenia sprawiły, że zacząłem lepiej rozumieć przekazywaną wiedzę i dostrzegać nowe możliwości, których nie zauważałem wcześniej. Dlatego mam zamiar zachęcić do podobnego podejścia – realizacji własnych zainteresowań poprzez samodzielne rozwiązywanie konkretnych problemów. Może się tutaj pojawić wiele wątpliwości: Od czego zacząć? Co robić, jeżeli nawet proste zadania sprawiają trudność? Na co się zdecydować, gdy jest tak wiele możliwości? Po co się czegoś

uczyć, skoro i tak być może nigdy się to nie przyda? To są pytania, które mnie także nurtowały. Brak odpowiedzi może natomiast prowadzić do zniechęcenia oraz utraty cennego czasu.

Umiejętności matematyczne i programistyczne są potrzebne przede wszystkim po to, by nauczyć się rozwiązywania niestandardowych problemów. Liczy się nie tyle wiedza sama w sobie, co proces jej zdobywania. Dlatego nie ma większego znaczenia to, od czego się zacznie i jaką drogę się wybierze – wystarczy regularnie poznawać kolejne zagadnienia w swoim tempie. Z czasem i tak decydujemy się na to, co nas interesuje, a przy okazji nabywamy umiejętności szybkiej nauki i radzenia sobie z nietypowymi sytuacjami w życiu. Można ten proces porównać do wspinania się na wysoką górę – zaczynamy dostrzegać krajobraz po drugiej stronie dopiero wtedy, gdy wykonamy pewien wysiłek. Trudno to zrozumieć na początku, gdy nie mamy jeszcze tego doświadczenia, jednak w pewnym momencie przychodzi chwila, gdy wszystko się niezwykle rozjaśnia – warto tego oczekiwać.

W tej książce staram się przedstawić materiał w takiej formie, w jakiej sam chciałbym go otrzymać – na podstawie wielu praktycznych przykładów. Każdy z nich zawiera:

- kod w **SageMath** z wynikiem lub wizualizacją;
- ręczne rozwiązanie (tam, gdzie jest to możliwe).

Według mnie jest to niezwykle ważne – przed napisaniem programu należy zrozumieć dane zagadnienie i umieć rozwiązać je samodzielnie na kartce. Zachęcam też do modyfikowania zaproponowanych przeze mnie rozwiązań – jest to najlepszy sposób na zapoznanie się z działaniem poszczególnych instrukcji.

Skrypt ten powstał z myślą o studentach kierunków ścisłych i stanowi uzupełnienie wykładu z algebry. Zakłada on znajomość podstaw matematyki oraz programowania na poziomie szkoły średniej. Wiele inspiracji do jego opracowania pochodzi z wykładów profesora Ryszarda Mazurka na Politechnice Białostockiej.

Rozpoczniemy od wprowadzenia do **SageMath** w celu zapoznania się z dostępnymi narzędziami. Następnie przejdziemy do rozdziału dotyczącego liczb zespolonych, gdzie zostanie przedstawiona niezbędna teoria potrzebna do zrozumienia tego tematu. Nowe narzędzia wprowadzane będą stopniowo, aby można było dostrzec ich związek z materiałem ze szkoły średniej, dotyczącym na przykład geometrii czy trygonometrii. Chciałbym skupić się na przedstawieniu różnorodnych pomysłów, które ułatwiały mi zrozumienie tych treści. Mam nadzieję, że każdy znajdzie tu coś dla siebie, nawet jeżeli są to tylko liczby zespolone.

Powodzenia! ☺

2. Wprowadzenie do SageMath

2.1. Środowisko CoCalc

SageMath to system algebry komputerowej, który umożliwia łatwiejsze wykonywanie zaawansowanych obliczeń matematycznych w wielu dziedzinach, takich jak analiza, algebra, analiza numeryczna, teoria grafów, kombinatoryka czy statystyka. Jest to wolne i otwarte oprogramowanie, będące alternatywą dla płatnych odpowiedników, m.in.: **Mathematica**, **MATLAB**, **Maple** czy **Magma**.

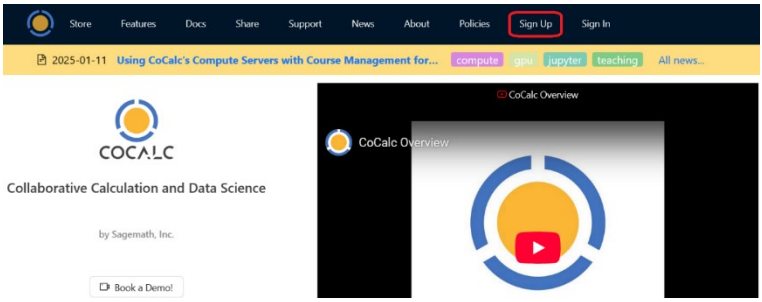
Język **Sage** opiera się na składni języka **Python**, jednak zawiera dodatkowo wiele wbudowanych funkcji, stałych i obiektów matematycznych. Dzięki temu nie ma potrzeby importowania ich z osobnych bibliotek. To sprawia, że nadal możemy stosować składnię **Pythona**, mając jednocześnie dostęp do wielu nowych obiektów, takich jak funkcje, wyrażenia symboliczne, równości czy wykresy.

W tej książce skupimy się na korzystaniu z **SageMath** w środowisku **CoCalc**[1]. Jest to platforma internetowa, która umożliwia pisanie kodu w języku **Sage** w notatnikach **Jupyter**. **SageMath** można również zainstalować lokalnie na własnym komputerze, jednak **CoCalc** oferuje wygodniejsze rozwiązanie, ponieważ nie wymaga konfiguracji środowiska.

2.1.1. Tworzenie pliku

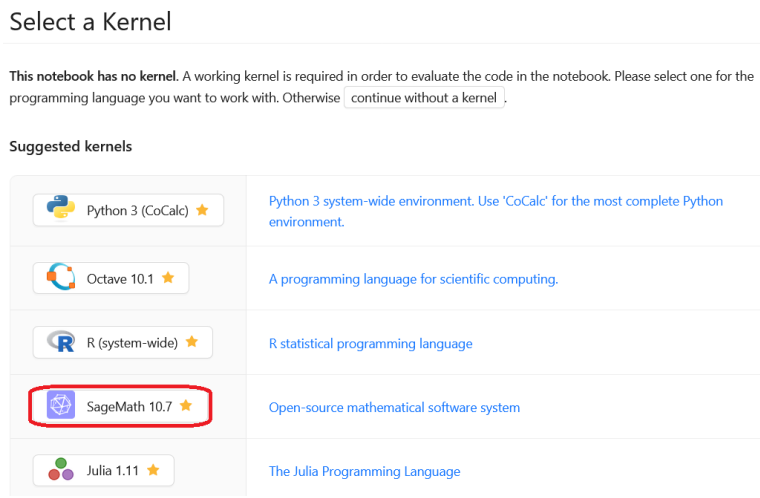
W celu rozpoczęcia pracy w **CoCalc** należy najpierw założyć darmowe konto, klikając przycisk **Sign Up** (zob. rysunek 2.1). Po rejestracji zostanie utworzony pusty projekt, w którym można dodawać nowe pliki lub przesyłać istniejące z własnego komputera.

[1] CoCalc, https://cocalc.com [dostęp: 18.10.2025].



Rysunek 2.1. Strona główna platformy **CoCalc**

Nowy notatnik tworzymy poprzez sekwencję poleceń: **New → Jupyter Notebook → SageMath**[2]. Na rysunku 2.2 pokazano ostatni krok – wybór kernela (jądra) **SageMath 10.7**. Kernel w notatniku **Jupyter** to silnik obliczeniowy odpowiedzialny za wykonywanie kodu w komórkach.



Rysunek 2.2. Wybór kernela **SageMath 10.7**

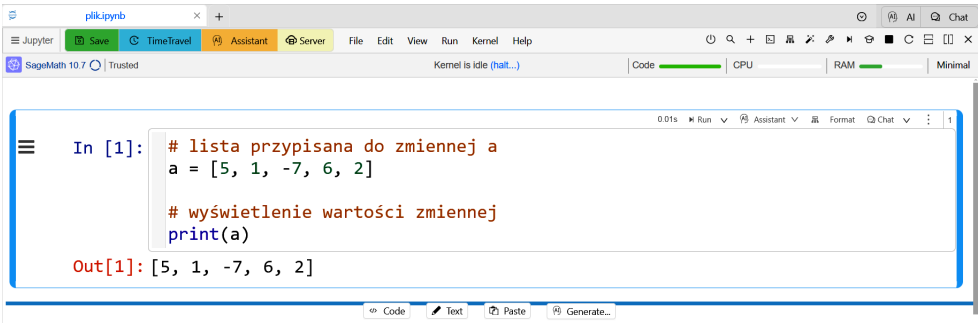
W ten sposób został utworzony nowy plik – notatnik **Jupyter** z rozszerzeniem **.ipynb**. Wprowadzone zmiany w środowisku **CoCalc** zapisywane są automatycznie. Ponadto dostępna jest funkcja **TimeTravel**, która umożliwia przywrócenie wcześniejszych wersji pliku. Notatnik można pobrać za pomocą ścieżki w menu: **File → Download File**. Warto również przechowywać pliki w formacie **.html**, który uzyskujemy poprzez: **File → Save and Export As HTML → HTML via Classic nbconvert (.html)**, ponieważ można je wtedy łatwo otworzyć w przeglądarce.

[2] Przycisk **New** wybieramy w widoku projektu. Po założeniu nowego konta należy ten krok pominąć.

2.1.2. Obsługa notatnika Jupyter

Nazwa **Jupyter** jest odniesieniem do trzech wspieranych języków programowania: **Julia**, **Python** oraz **R**. Główną zaletą notatników **Jupyter** jest możliwość wykonywania pojedynczych fragmentów kodu bez konieczności pisania całego programu. Poszczególne komórki (**Cells**) uruchamiamy za pomocą skrótu klawiszowego **Ctrl** + **Enter** lub przycisku **Run** umieszczonego w prawym górnym rogu.

Nowy plik zawiera pustą komórkę kodową (**Code Cell**). Możemy w niej umieścić dowolny kod w języku **Python** lub **Sage** (zob. przykład na rysunku 2.3). Po uruchomieniu komórki kodowej po jej lewej stronie pojawia się numer wskazujący kolejność wykonania poszczególnych bloków kodu w notatniku.



Rysunek 2.3. Komórka kodowa zawierająca przykładowy kod w języku **Python**

Pod każdą komórką znajdują się przyciski pozwalające dodać nową komórkę kodową lub komórkę tekstową (**Text Cell**). Komórki tekstowe nie są numerowane. Służą one do przechowywania zwykłego tekstu, tekstu sformatowanego za pomocą składni **Markdown** oraz wzorów matematycznych zapisanych w języku **LaTeX**.

W kolejnych blokach możemy korzystać z kodu zawartego w uruchomionych komórkach. Na przykład poniżej definiujemy nową zmienną b, pisząc:

```
b = 12
```

W następnych komórkach można odwoływać się do wartości przypisanej do zmiennej b. Przykładowo wypisujemy tę wartość na ekran za pomocą funkcji **print()**:

```
print(b)
Out: 12
```

Ponadto w notatnikach **Jupyter** ostatnia instrukcja w komórce jest wyświetlana na ekranie automatycznie, dlatego w poprzednim poleceniu nie ma potrzeby korzystania z funkcji **print()**, gdyż ten sam efekt można osiągnąć za pomocą prostszego zapisu:

```
b
Out: 12
```

2.1.3. Skróty klawiszowe

W tabeli 2.1 przedstawiono najczęściej używane skróty klawiszowe, które możemy stosować w notatniku **Jupyter**:

Tabela 2.1. Wybrane skróty klawiszowe w notatniku **Jupyter**

	Skrót	Opis
Uruchamianie	**Ctrl + Enter**	Uruchamia komórkę i pozostaje w niej
	Shift + Enter	Uruchamia komórkę i przechodzi do następnej
	Alt + Enter	Uruchamia komórkę i tworzy nową poniżej
Nawigacja	**Esc**	Wyjście z komórki (powoduje jej podświetlenie na niebiesko)
	Enter	Wejście do komórki (operacja odwrotna do **Esc**)
	Shift + Tab	Wyświetla dokumentację funkcji
Edycja	**Tab**	Autouzupełnienie kodu (po wpisaniu początku nazwy wyświetlane są propozycje)
	Ctrl + /	Zakomentowanie/odkomentowanie zaznaczonego bloku kodu
	Ctrl + LPM	Umieszczenie dodatkowego kursora w miejscu wskazanym lewym przyciskiem myszy
Tworzenie i usuwanie	**a**	Tworzy nową, pustą komórkę nad bieżącą (**Above**)
	b	Tworzy nową, pustą komórkę pod bieżącą (**Below**)
	d d (lub x)	Usuwa bieżącą komórkę (**Delete**)
Modyfikacja i przerywanie	**m**	Zmienia typ komórki na tekstową (**Markdown**)
	y	Zmienia typ komórki na kodową (**Code**)
	i i	Przerywa wykonywanie komórki (**Interrupt**)

Uwaga. Skróty występujące w ostatnich sześciu wierszach tabeli wymagają przejścia w tzw. tryb poleceń (uzyskujemy go poprzez niebieskie podświetlenie komórki po naciśnięciu klawisza **Esc** lub kliknięciu w jej lewą część). Są one szczególnie przydatne podczas szybkiego formatowania notatnika bez użycia myszy.

2.2. Język Sage

Przejdziemy teraz do prezentacji obiektów i funkcji wbudowanych dostępnych w języku **Sage**. Przy poznawaniu nowego narzędzia warto najpierw ogólnie zorientować się w jego podstawowych możliwościach. Taka znajomość dostępnych opcji sprawia, że podczas rozwiązywania praktycznych problemów można pracować swobodniej i efektywniej.

Język **Sage** bazuje na **Pythonie**, który cechuje się dużą elastycznością i pozwala na różne, często kreatywne podejścia do tego samego zagadnienia. Wynika to z dostępności wielu intuicyjnych elementów składniowych oraz przydatnych funkcji wbudowanych, które można łączyć na różne sposoby. Dzięki temu możemy w sposób zwięzły pisać kod, który jednocześnie pozostaje czytelny i przejrzysty.

W tej książce skupimy się na zapoznaniu ze składnią języka **Sage**, bez zagłębiania się w szczegóły **Pythona**. Gdy zajdzie potrzeba użycia jakiegoś elementu jego składni, zostanie to zilustrowane odpowiednim przykładem. Zaprezentuje on sposób działania w odniesieniu do konkretnego zagadnienia matematycznego.

2.2.1. Wywoływanie funkcji

W języku **Python** wyróżniamy dwie podstawowe konwencje stosowania funkcji. Możemy tę czynność wykonać poprzez:

1. Wywołanie bezpośrednie.

Zaczynamy od nazwy funkcji, wykorzystując składnię:

funkcja(*zmienna*, *arg1*, *arg2*, *arg3*, …).

W tym przypadku zmienna jest pierwszym argumentem funkcji **funkcja()**, po którym umieszczamy kolejne argumenty (jednak mogą one w ogóle nie występować). Na przykład funkcja **round()** zwraca liczbę zaokrągloną do określonej liczby miejsc po przecinku:

```
z = 3.14159
round(z, 2)
Out: 3.14
```

2. Operator kropki.

Rozpoczynamy od zmiennej, na której stosujemy funkcję, zgodnie ze składnią:

zmienna.**funkcja**(*arg1*, *arg2*, *arg3*, …).

Tym razem wykorzystujemy operator kropki, aby zasygnalizować użycie funkcji **funkcja()**. Pozostałe argumenty (które także mogą nie występować) podajemy w nawiasie. Na przykład funkcja **nth_root()** zwraca pierwiastek stopnia przekazanego jako pierwszy argument:

```
z.nth_root(3)
Out: 1.46459147519879
```

Ponadto w drugiej konwencji funkcje możemy łączyć ze sobą w jednym ciągu, stosując kolejne operatory kropki. W ten sposób kod

```
z.nth_root(3).round()
Out: 1
```

najpierw oblicza pierwiastek trzeciego stopnia z liczby z, a następnie zwraca wynik zaokrąglony do liczby całkowitej.

2.2.2. Funkcje i stałe matematyczne

W języku **Sage** domyślnie stosowane są obliczenia na symbolach. W przeciwieństwie do obliczeń numerycznych, które zwracają wyniki przybliżone, obliczenia symboliczne:

- zwracają wyniki dokładne;
- pozwalają na manipulację wzorami i wyrażeniami jak w klasycznej matematyce;
- działają na zmiennych i wyrażeniach bez konieczności przypisywania im konkretnych wartości liczbowych.

Zobaczymy to na przykładzie funkcji **sqrt()**, która zwraca pierwiastek kwadratowy z danej liczby:

```
sqrt(12)
Out: 2*sqrt(3)
```

Widzimy, że wynik został przedstawiony w postaci symbolicznej, a nie przybliżonej. Aby uzyskać przybliżenie numeryczne, stosujemy funkcję **n()**.

Przykład 2.1
Znajdziemy przybliżenie numeryczne liczby ln 7.

Aby uzyskać logarytm naturalny, korzystamy z funkcji **ln()** lub **log()**. W przypadku funkcji **n()**, zwracającej przybliżenie numeryczne, dopuszczalne są obie konwencje stosowania funkcji. Zaczniemy od wywołania bezpośredniego:

```
n( ln(7) )
Out: 1.94591014905531
```

Na ogół wygodniejsze będzie jednak wykorzystanie operatora kropki – w ten sposób możemy łączyć ze sobą różne funkcje, co poprawia czytelność przy wykonywaniu bardziej złożonych operacji:

```
ln(7).n()
Out: 1.94591014905531
```

Automatycznie wbudowane są również najczęściej używane stałe matematyczne, czyli m.in.: stała π, liczba Eulera e oraz jednostka urojona i:

```
print( cos(pi) )
print( log(e) )
print( i^2 )
Out: -1
 1
 -1
```

Uwaga. W **Sage** potęgowanie możemy zapisywać za pomocą symbolu ^, który w **Pythonie** oznacza inną operację – alternatywę wykluczającą[3].

[3] Zapis i**2 stosowany w języku **Python** również jest poprawny.

W tabeli 2.2 przedstawiono zestawienie najczęściej używanych funkcji matematycznych:

Tabela 2.2. Przykładowe funkcje matematyczne dostępne w języku **Sage**

	Nazwa funkcji	Przykład	Zapis matematyczny
Funkcje trygonometryczne	**sin()**	sin(pi/3)	$\sin\frac{\pi}{3}=\frac{\sqrt{3}}{2}$
	cos()	cos(pi/3)	$\cos\frac{\pi}{3}=\frac{1}{2}$
	tan()	tan(pi/3)	$\mathrm{tg}\frac{\pi}{3}=\sqrt{3}$
	cot()	cot(pi/3)	$\mathrm{ctg}\frac{\pi}{3}=\frac{\sqrt{3}}{3}$
Funkcje cyklometryczne	**asin()**	asin(1)	$\arcsin 1=\frac{\pi}{2}$
	acos()	acos(1)	$\arccos 1=0$
	atan()	atan(1)	$\mathrm{arctg}\,1=\frac{\pi}{4}$
	acot()	acot(1)	$\mathrm{arcctg}\,1=\frac{\pi}{4}$
Logarytm i eksponenta	**log()**	log(32, 2)	$\log_2 32=5$
	exp()	exp(i*pi)	$e^{i\pi}=-1$
Podłoga i sufit	**floor()**	floor(3.14)	$\lfloor 3{,}14\rfloor=3$
	ceil()	ceil(3.14)	$\lceil 3{,}14\rceil=4$
NWD i NWW	**gcd()**	gcd(6, 15)	$\mathrm{NWD}(6,15)=3$
	lcm()	lcm(6, 15)	$\mathrm{NWW}(6,15)=30$
Silnia i symbol Newtona	**factorial()**	factorial(5)	$5!=120$
	binomial()	binomial(4, 2)	$\binom{4}{2}=6$
Dzielniki i rozkład na czynniki pierwsze	**divisors()**	divisors(12)	$\mathrm{D}_{12}=\{1,2,3,4,6,12\}$
	factor()	factor(2024)	$2024=2^3\cdot 11\cdot 23$

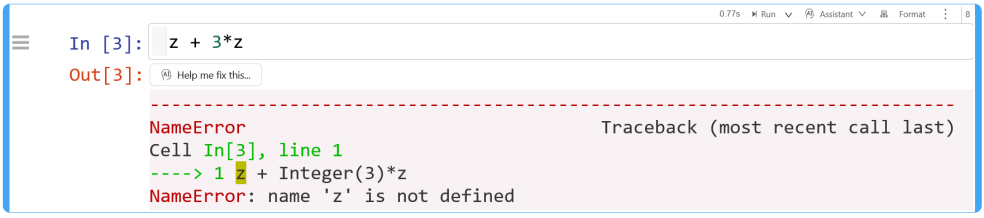
2.2.3. Wyrażenia algebraiczne

System **SageMath** pozwala tworzyć wyrażenia symboliczne, operować na zmiennych bez przypisywania im konkretnych wartości liczbowych oraz przeprowadzać dokładne obliczenia. Zmienna x traktowana jest jako symbol domyślnie, co pozwala od razu wykonywać działania na wyrażeniach algebraicznych:

```
(2*x)^2 + x + 3*x + 1
Out: 4*x^2 + 4*x + 1
```

Gdy jednak chcemy użyć innej nazwy zmiennej, należy ją najpierw zadeklarować. W przeciwnym przypadku pojawi się następujący błąd:

```
z + 3*z
NameError: name 'z' is not defined
```



Rysunek 2.4. Komunikat o błędzie w notatniku **Jupyter**

Podczas pisania kodu komunikaty o błędach pojawiają się bardzo często, dlatego warto nauczyć się właściwie je interpretować. Powyższy błąd przedstawiono na rysunku 2.4. Najważniejsza informacja znajduje się na samym dole – wskazuje ona, co należy poprawić. W naszym przypadku próbowaliśmy wykonać działanie na wyrażeniach algebraicznych, ale nie zdefiniowaliśmy zmiennej z jako symbolu. Trzeba to wykonać za pomocą funkcji **var()**:

```
var('z')
z + 3*z
Out: 4*z
```

Możemy również deklarować jako symbole wiele zmiennych jednocześnie oraz przypisywać wyrażenia algebraiczne do zmiennych i wykonywać na nich działania:

```
var('s t u')
wyrazenie = -s + 3*t + 2*u
wyrazenie + 2*s
Out: s + 3*t + 2*u
```

W **Sage** występuje rozróżnienie między wyrażeniem a funkcją. Przeanalizujemy główne różnice w ich zachowaniu, ponieważ może to być później mylące podczas rozwiązywania trudniejszych problemów:

1. **Wyrażenia** definiujemy poprzez przypisanie do zmiennej wyrażenia algebraicznego. Na początku wszystkie zmienne występujące w tym wyrażeniu deklarujemy jako symbole, korzystając z funkcji **var()**:

```
var('n')
W = n^3 + 2*n^2 - 7
```

Aby obliczyć wartość wyrażenia `W` dla konkretnej wartości zmiennej, na przykład $n = 2$, należy koniecznie zastosować zapis:

wyrażenie(*zmienna=wartość*).

Jest to najistotniejsza różnica między wyrażeniem a funkcją. W naszym przykładzie napiszemy:

```
W(n=2)
Out: 9
```

Dla wyrażeń złożonych z wielu zmiennych wykorzystujemy analogiczną składnię, czyli:

```
var('y')
G = 2*x - 3*y
G(x=2, y=1)
Out: 1
```

2. **Funkcje** definiujemy poprzez umieszczenie po nazwie funkcji nawiasów, w których określamy zmienne zależne. Wszystkie te zmienne zostaną od razu potraktowane jako symbole, więc tym razem nie ma konieczności stosowania funkcji **var()**:

```
f(x) = x^2 + 2*x - 3
```

W przeciwieństwie do wyrażenia przy obliczaniu wartości funkcji dla danego argumentu, na przykład $x = 3$, nie musimy stosować zapisu ze znakiem równości (=). Wystarczy przekazać poszczególne argumenty w kolejności odpowiadającej zmiennym w definicji funkcji:

```
f(3)
Out: 12
```

W podobny sposób można tworzyć złożenia funkcji, na przykład $f(2x)$:

```
f(2*x)
Out: 4*x^2 + 4*x - 3
```

Dodatkowo funkcja, w odróżnieniu od wyrażenia, jest wyświetlana razem ze strzałką:

```
f
Out: x |--> x^2 + 2*x - 3
```

Przeanalizujemy teraz najczęściej występujące błędy, które popełniane są podczas nauki języka **Sage**:

1. **Brak operatora gwiazdki przy mnożeniu** – należy stosować zapis `3*x`, `5*x^2` itd. Instrukcja `2x` powoduje błąd `invalid decimal literal`. Wobec tego, jeśli pojawia się taki komunikat, należy szukać w kodzie brakujących gwiazdek:

```
2x
SyntaxError: invalid decimal literal
```

2. **Brak deklaracji zmiennych jako symboli** – powoduje błąd `name is not defined`, z którym już wcześniej się spotkaliśmy. Jeżeli przypiszemy do zmiennej jakąś wartość liczbową, na przykład `x = 7`, to od tej pory będzie ona traktowana

jako liczba, a nie symbol. Aby ponownie wykonywać działania na wyrażeniach algebraicznych, należy zastosować polecenie var('x'):

```
x = 7
x + 2*x
Out: 21
```

3. **Stosowanie nazw dostępnych w Sage** – nadpisanie wbudowanych nazw, na przykład poprzez cos = 1.25 lub e = 2, sprawia, że dana funkcja wbudowana lub stała (tutaj cosinus i liczba Eulera) będzie odtąd traktowana jak zwykła liczba. W rezultacie późniejsze użycie tych nazw spowoduje błąd object is not callable lub błędny wynik:

```
cos = 1.25
cos(0)
TypeError: 'sage.rings.real_mpfr.RealLiteral' object is not callable
```

Ten efekt uboczny można naprawić poprzez restart kernela, który uzyskujemy za pomocą ścieżki w menu: **Kernel → Restart Kernel** (jest to dobre rozwiązanie w wielu sytuacjach, kiedy coś nie działa tak, jak powinno) lub korzystając z polecenia reset().

4. **Brak nawiasów, gdy istotna jest kolejność działań** – wyrażenia w liczniku i mianowniku ułamka są traktowane, jakby były w nawiasach. Dlatego podczas definiowania ułamka

$$\frac{x+2}{x-1}$$

należy koniecznie dodać w kodzie nawiasy:

```
wyrazenie = (x + 2) / (x - 1)
```

Jeżeli napiszemy: x + 2 / x - 1, to obowiązywać będzie kolejność działań, więc w wyniku otrzymamy błędny rezultat:

$$x + \frac{2}{x} - 1.$$

Wyrażenia algebraiczne również domyślnie zapisywane są w najprostszej postaci. Możemy jednak wymusić określone zachowanie wyrażeń za pomocą odpowiednich funkcji:

- **expand()** – zapisuje wyrażenie w postaci ogólnej:

```
W = (x+2)^3
W.expand()
Out: x^3 + 6*x^2 + 12*x + 8
```

- **factor()** – zapisuje wyrażenie w postaci iloczynowej:

```
W = x^2 - x - 6
W.factor()
Out: (x + 2)*(x - 3)
```

- **simplify()** – upraszcza wyrażenie (w zależności od jego rodzaju możemy stosować funkcje, takie jak **simplify_rational()** dla wyrażeń wymiernych, **simplify_trig()** dla funkcji trygonometrycznych lub **simplify_full()** w celu ogólnego uproszczenia):

```
W = (x^3 - 3*x - 2) / (x^2 - x - 2)
W.simplify_rational()
Out: x + 1
```

Obiekty w **Sage** możemy wyświetlać w bardziej przejrzystej formie za pomocą funkcji **show()**:

```
show( sqrt(x) + 1/x )
Out: √x + 1/x
```

Funkcja **LatexExpr()** pozwala natomiast przedstawiać wyniki w eleganckiej formie, z wykorzystaniem składni języka **LaTeX**:

```
var('a b c')
delta = b^2 - 4*a*c
```

```
print('Wyróżnik trójmianu kwadratowego:')
show( LatexExpr(r'\Delta='), delta )
Out: Δ = b^2 − 4ac
```

Uwaga. Litera r przed tekstem zapisanym w języku **LaTeX** powoduje, że symbol backslash (\) jest traktowany dosłownie, a nie jako znak specjalny[4].

Przykład 2.2

Dany jest wielomian:

$$W(x) = x^4 + x^3 - 7x^2 - x + 6.$$

a) Zapiszemy wielomian W w postaci iloczynowej.

b) Podamy dokładną wartość wielomianu dla argumentu $x = \frac{1}{2}$.

c) Wyznaczymy przybliżoną wartość wielomianu dla $x = \sqrt{2}$.

d) Przedstawimy iloczyn $W(x) \cdot (x + 1)$ w postaci ogólnej.

Wielomian W definiujemy jako funkcję – ten zapis jest na ogół wygodniejszy ze względu na prostszą składnię. Poniżej stosujemy typową konwencję wykorzystywaną w notatnikach **Jupyter** – najpierw definiujemy obiekt, a następnie go wyświetlamy, aby zobaczyć, jak wygląda:

```
W(x) = x^4 + x^3 - 7*x^2 - x + 6
W
Out: x |--> x^4 + x^3 - 7*x^2 - x + 6
```

a) Zapisujemy wielomian W w postaci iloczynowej, korzystając z funkcji **factor()**. Stosujemy tutaj operator kropki:

```
W.factor()
Out: (x + 3)*(x + 1)*(x - 1)*(x - 2)
```

[4] W języku **Python** zapis \n oznacza symbol nowej linii, natomiast \t – tabulator. Zamiast tzw. surowego tekstu uzyskanego poprzez wykorzystanie litery r można też stosować podwójny backslash (\\).

b) Ponieważ zdefiniowaliśmy wielomian jako funkcję, nie musimy precyzować zmiennej przy obliczaniu wartości:

```
W(1/2)
Out: 63/16
```

c) Aby znaleźć przybliżoną wartość wielomianu W dla argumentu $x = \sqrt{2}$, korzystamy z funkcji **n()**:

```
W( sqrt(2) ).n()
Out: -2.58578643762691
```

d) Iloczyn $W(x) \cdot (x + 1)$ przedstawiamy w postaci ogólnej za pomocą funkcji **expand()**. Tutaj wykorzystujemy wywołanie bezpośrednie:

```
expand( W * (x+1) )
Out: x |--> x^5 + 2*x^4 - 6*x^3 - 8*x^2 + 5*x + 6
```

Uwaga. Uzyskany wynik jest funkcją, natomiast zapis `expand( W(x) * (x+1) )` spowoduje, że w wyniku otrzymamy wyrażenie.

Warto zajrzeć do dokumentacji wyrażeń symbolicznych[5] w **Sage** [4], aby poznać więcej dostępnych możliwości dotyczących obliczeń na symbolach.

2.3. Rysunki

System **SageMath** oferuje intuicyjną składnię do tworzenia rysunków matematycznych. Poszczególne elementy graficzne można w prosty sposób ze sobą łączyć oraz modyfikować. Umożliwia to tworzenie złożonych ilustracji bez konieczności pisania dużej ilości kodu – w przeciwieństwie do wielu innych narzędzi, takich jak biblioteki języka **Python**.

W **Sage** można wykonywać wykresy dwuwymiarowe (2D) oraz interaktywne wykresy trójwymiarowe (3D). W tej części skupimy się na omówieniu najczęściej używanych typów wykresów 2D oraz argumentów, które możemy w nich stosować. Będą one szczególnie przydatne przy wizualizacji pojęć związanych z liczbami zespolonymi.

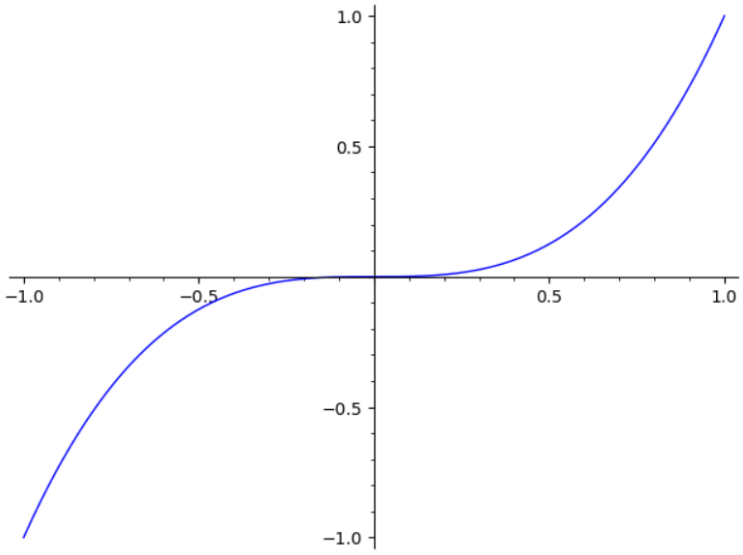
[5] Sage, *Symbolic expressions*, https://doc.sagemath.org/html/en/reference/calculus/sage/symbolic/expression.html [dostęp: 18.10.2025].

2.3.1. Rodzaje funkcji graficznych

Przeanalizujemy podstawowe funkcje dostępne w **Sage** służące do generowania wykresów funkcji oraz rysowania figur geometrycznych:

1. Wykres funkcji – plot().
Aby narysować wykres, jako pierwszy argument funkcji **plot()** przekazujemy funkcję, wyrażenie lub wzór funkcji, której wykres chcemy otrzymać. Domyślnie dziedziną jest przedział $(-1, 1)$:

```
plot(x^3)
```



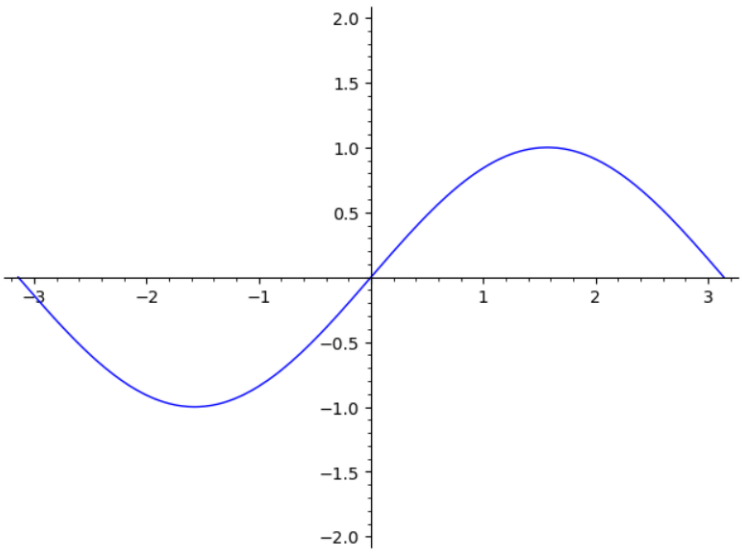
Rysunek 2.5. Prosty wykres funkcji $f(x) = x^3$

Uzyskany wykres przedstawiono na rysunku 2.5. Alternatywnie możemy najpierw zdefiniować funkcję f(x) = x^3, a następnie otrzymać ten sam efekt za pomocą polecenia plot(f). Aby zmienić dziedzinę, piszemy:

```
plot( x^3, (x, -2, 2) )
```

Zbiór wyświetlanych wartości można kontrolować za pomocą argumentów ymin i ymax. Przykładowo narysujemy wykres funkcji $f(x) = \sin x$ dla argumentów z przedziału $(-\pi, \pi)$ z wartościami ograniczonymi do przedziału $(-2, 2)$:

```
plot( sin(x), (x, -pi, pi), ymin=-2, ymax=2 )
```

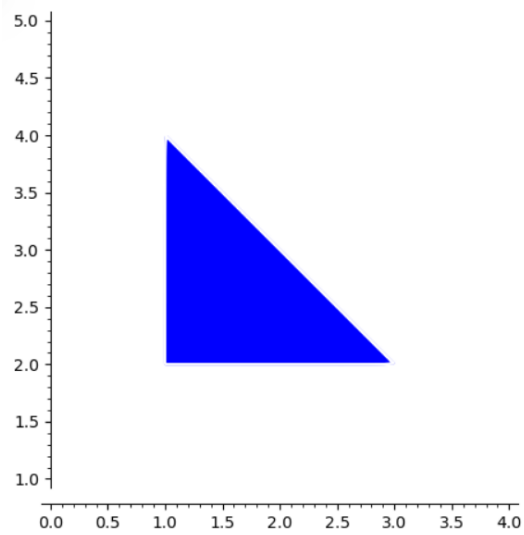


Rysunek 2.6. Wykres funkcji $f(x) = \sin x$ z określoną dziedziną oraz zakresem wyświetlanych wartości

Wynik działania kodu pokazano na rysunku 2.6. Warto zwrócić uwagę, że pierwsze dwa argumenty (wzór funkcji i dziedzinę) podajemy bez znaku równości – są to tzw. argumenty pozycyjne. Funkcja **plot()** wymaga, aby pierwszy argument określał wzór funkcji, a drugi – dziedzinę, co eliminuje potrzebę podawania ich nazw. Dwa ostatnie argumenty mają natomiast konkretne nazwy: ymin i ymax – są to tzw. argumenty nazwane – można je podawać w dowolnej kolejności.

2. Obszar – region_plot().
Funkcja **region_plot()** służy do zaznaczania obszaru na płaszczyźnie, określonego za pomocą ograniczeń podanych w nawiasach kwadratowych. Warunki oddzielone przecinkami oznaczają ich część wspólną:

```
var('x y')
region_plot( [x > 1, y > 2, y < 5-x], (x, 0, 4), (y, 1, 5) )
```



Rysunek 2.7. Trójkąt ograniczony prostymi $x = 1$, $y = 2$ i $y = 5 - x$, uzyskany za pomocą funkcji **region_plot()**

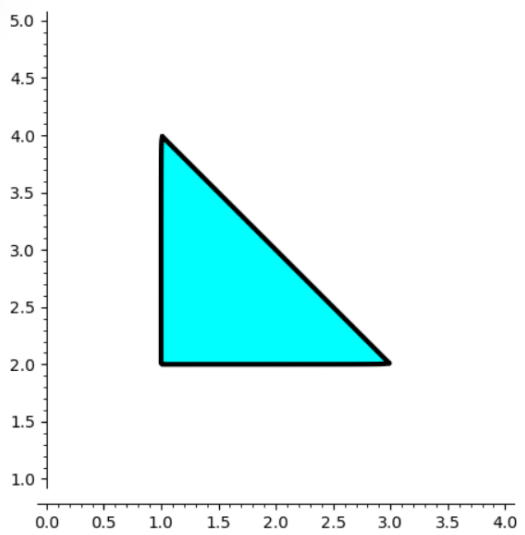
Otrzymaną figurę przedstawiono na rysunku 2.7. Pierwszym argumentem funkcji **region_plot()** jest lista zawierająca warunki oddzielone przecinkami – w tym przypadku jest to część wspólna obszarów, które znajdują się:

- po prawej stronie prostej pionowej: $x = 1$;
- nad prostą poziomą: $y = 2$;
- pod prostą ukośną: $y = 5 - x$.

Następnie określamy zakresy: przedział $(0, 4)$ dla zmiennej x (oś pozioma) oraz przedział $(1, 5)$ dla zmiennej y (oś pionowa). Ponieważ wykresy generowane są na podstawie zmiennych symbolicznych, konieczne jest zadeklarowanie zmiennych x i y jako symboli za pomocą funkcji **var()**.

Początkowo uzyskany obszar nie prezentuje się jeszcze zbyt estetycznie, dlatego warto poprawić jego wygląd, modyfikując kolory brzegu i wypełnienia oraz grubość linii poprzez zmianę odpowiednich argumentów:

```
region_plot( [x > 1, y > 2, y < 5-x], (x, 0, 4), (y, 1, 5),
 incol='cyan', bordercol='black', borderwidth=3 )
```



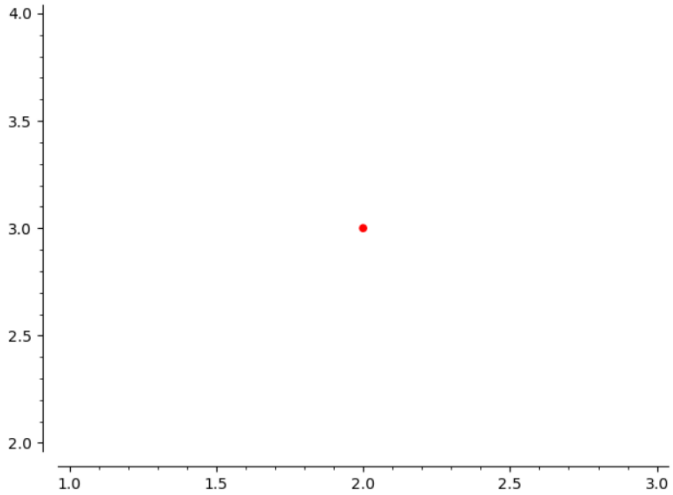
Rysunek 2.8. Efekt działania funkcji **region_plot()** po zmianie kolorów wnętrza i brzegu oraz grubości linii

Wynikowy obszar przedstawiono na rysunku 2.8. Argumenty incol oraz bordercol odpowiadają za kolor wypełnienia i obramowania, natomiast borderwidth ustala grubość linii brzegowej. Aby określić grubość brzegu, należy najpierw zdefiniować jego kolor – dlatego usunięcie argumentu bordercol='black' w powyższym kodzie spowoduje błąd. W przypadku użycia wielu argumentów warto podzielić kod na kilka linijek w celu zwiększenia jego czytelności.

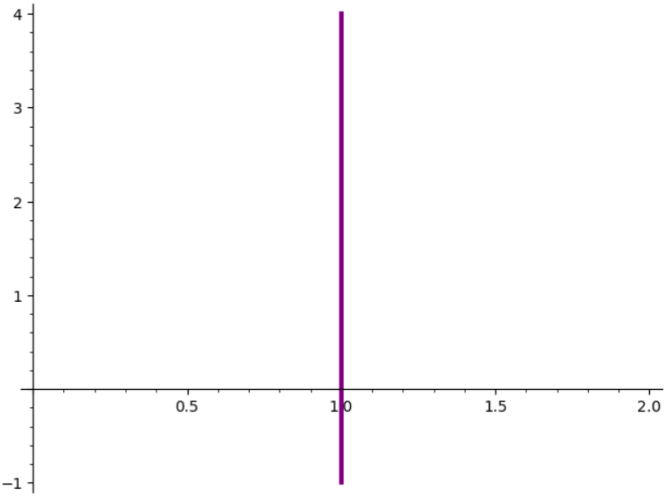
3. Punkt – point().
Aby narysować punkt na płaszczyźnie, należy przekazać jego współrzędne do funkcji **point()**. Podobnie jak inne funkcje graficzne **point()** posiada własny zestaw argumentów. Poniżej znajduje się kod, który sprawdza się dla większości rysunków:

```
point( (2, 3), color='red', size=30, zorder=5 )
```

Efekt działania pokazano na rysunku 2.9. Wykorzystujemy tutaj standardowy zestaw argumentów, który warto stosować dla każdego punktu. Argumenty color i size zmieniają odpowiednio kolor i rozmiar punktu, natomiast zorder określa położenie punktu względem innych figur (większa wartość oznacza, że punkt znajduje się bardziej z przodu) – jest to potrzebne przy umieszczaniu na rysunku większej liczby obiektów.



Rysunek 2.9. Punkt na płaszczyźnie o współrzędnych $(2, 3)$ wygenerowany za pomocą funkcji **point()**



Rysunek 2.10. Odcinek łączący punkty o współrzędnych $(1, -1)$ oraz $(1, 4)$ uzyskany za pomocą funkcji **line()**

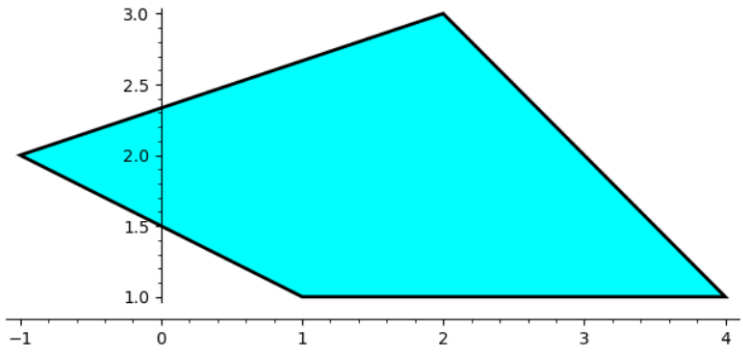
4. Linia – line().
Aby narysować odcinek, umieszczamy w liście współrzędne jego końców:

```
line( [(1, -1), (1, 4)], color='purple', thickness=3 )
```

Uzyskaną linię przedstawiono na rysunku 2.10. Funkcja **line()** przyjmuje jako pierwszy argument listę zawierającą współrzędne punktów, które chcemy połączyć. Ta funkcja także ma własny zestaw argumentów – tym razem do określenia grubości linii służy argument thickness. Warto zwrócić uwagę, że linie, które nie są pionowe, można również rysować za pomocą funkcji **plot()**, podając wzór odpowiedniej funkcji.

5. Wielokąt – polygon().
Pierwszym argumentem funkcji **polygon()** jest lista zawierająca współrzędne kolejnych wierzchołków wielokąta. Ważne jest przekazanie ich w odpowiedniej kolejności – wzdłuż boków wielokąta:

```
polygon( [(1, 1), (-1, 2), (2, 3), (4, 1)],
 color='cyan', edgecolor='black', thickness=2 )
```



Rysunek 2.11. Wielokąt utworzony za pomocą funkcji **polygon()**

Na rysunku 2.11 przedstawiono wynik działania funkcji. Pierwszy argument jest listą zawierającą wierzchołki podane w kolejności zgodnej z ruchem wskazówek zegara. Kolejne argumenty – color, edgecolor i thickness – formatują odpowiednio kolor wnętrza, kolor krawędzi i grubość linii obramowania. Aby otrzymać wielokąt bez wypełnienia, należy dodać parametr fill=False. W przypadku podania

nieprawidłowej nazwy argumentu **SageMath** wyświetli listę podpowiedzi z dostępnymi nazwami.

2.3.2. Argumenty funkcji kreślenia

Wygląd wykresu modyfikujemy poprzez użycie odpowiednich argumentów. Przyjrzymy się teraz argumentom, które dostępne są w najczęściej używanej funkcji graficznej – **plot()**:

1. Kolor wykresu – color.
Listę nazwanych kolorów można znaleźć w dokumentacji biblioteki **Matplotlib**[6], z której system **SageMath** korzysta do generowania wykresów 2D. Możemy też wybrać dowolny kolor za pomocą modelu **RGB**. W tym celu przekazujemy jako argument odpowiedni kod szesnastkowy. Poniżej zmieniamy kolor paraboli $y = x^2$ na malinowy:

```
plot(x^2, color='#d43b64')
```

2. Styl linii – linestyle.
Mamy do wyboru następujące style linii:

- '-' – linia ciągła (argument domyślny);
- '--' – linia przerywana;
- ':' – linia kropkowana;
- '-.' – linia kreskowo-kropkowana;
- '' – brak linii.

Poniższy kod pozwala naszkicować wykres funkcji $y = x^2$ linią przerywaną:

```
plot(x^2, linestyle='--')
```

3. Styl punktu – marker.
Proces tworzenia wykresów w **SageMath** składa się z dwóch etapów:

- najpierw generowane są punkty (argumenty *x* wybierane są losowo z dziedziny, natomiast wartości *y* są obliczane na podstawie przekazanego wzoru);
- następnie uzyskane punkty łączone są linią ciągłą.

Styl punktów można wybierać spośród wielu dostępnych możliwości, m.in.: punkt ('.'), okrąg ('o'), kwadrat ('s'), trójkąt ('v'), plus ('+') itd. Za pomocą

[6] Matplotlib, *List of named colors*, https://matplotlib.org/stable/gallery/color/named_colors.html [dostęp: 18.10.2025].

argumentów markersize, markeredgecolor i markerfacecolor zmieniamy odpowiednio rozmiar punktu oraz kolor jego obramowania i wypełnienia.

Pozostałe istotne argumenty dostępne dla funkcji **plot()** zostały zebrane w tabeli 2.3.

Tabela 2.3. Argumenty funkcji **plot()**

	Argument	**Przykład zastosowania**
Grubość linii	thickness	plot(x^2, thickness=3)
Tytuł wykresu	title	plot(x^2, title='Parabola')
Przezroczystość	alpha	plot(x^2, alpha=0.6)
Linie siatki	gridlines	plot(x, gridlines=True)
Etykieta legendy	legend_label	plot(x, legend_label='Prosta')
Rozmiar wykresu	figsize	plot(x, figsize=7)

Przykład 2.3

Narysujemy wykres wielomianu danego wzorem:

$$W(x) = x^3 - 2x^2 - 3x + 4,$$

dla argumentów z przedziału $(-3, 4)$, a następnie:
a) ustalimy zbiór wyświetlanych wartości jako $(-8, 8)$;
b) zmienimy kolor wykresu i grubość linii;
c) dodamy tytuł, legendę oraz linie siatki.

Najpierw definiujemy wielomian W i generujemy jego wykres za pomocą funkcji **plot()**. Wykresy domyślnie rysowane są w zakresie $(-1, 1)$, dlatego zgodnie z poleceniem zmieniamy dziedzinę na przedział $(-3, 4)$:

```
W(x) = x^3 - 2*x^2 - 3*x + 4
plot( W, (x, -3, 4) )
```

a) Aby lepiej zilustrować zachowanie funkcji, ograniczamy zakres wyświetlanych wartości, dodając do funkcji **plot()** argumenty ymin i ymax:

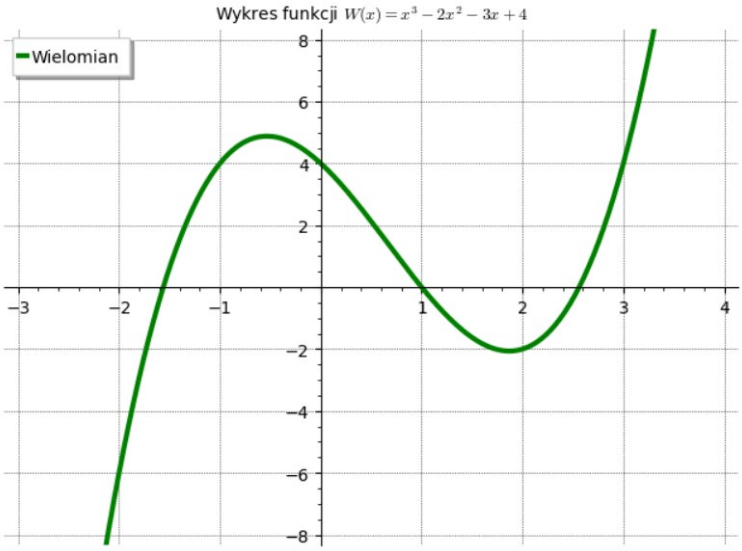
```
plot( W, (x, -3, 4), ymin=-8, ymax=8 )
```

b) Następnie poprawiamy czytelność wykresu poprzez zmianę jego koloru na zielony i zwiększenie grubości linii. Bardziej złożone rysunki warto tworzyć

stopniowo – dodajemy kolejne elementy krok po kroku i obserwujemy, jak zmiany wpływają na wygląd rysunku. Nie należy pisać od razu całego kodu – warto podzielić ten proces na mniejsze etapy.

```
plot( W, (x, -3, 4), ymin=-8, ymax=8,
 color='green', thickness=3 )
```

c) Na koniec dodajemy pozostałe argumenty, które zostały opisane w tabeli 2.3. Umieszczenie fragmentu tekstu pomiędzy „dolarami" ($$) pozwala korzystać ze składni języka **LaTeX**. Efekt końcowy, uzyskany za pomocą poniższego kodu, przedstawiono na rysunku 2.12.



Rysunek 2.12. Końcowy wykres wielomianu $W(x) = x^3 - 2x^2 - 3x + 4$

```
plot( W, (x, -3, 4), ymin=-8, ymax=8, color='green',
 thickness=3, legend_label='Wielomian', gridlines=True,
 title='Wykres funkcji $W(x)=x^3-2x^2-3x+4$' )
```

W podobny sposób możemy zaznaczać na rysunku inne figury. Przykładowo poniższy kod generuje okrąg jednostkowy o środku w początku układu współrzędnych:

```
circle( (0, 0), 1 )
```

Pozostałe funkcje graficzne i wiele innych dostępnych możliwości można znaleźć w dokumentacji wykresów 2D[7] w **Sage**.

2.3.3. Łączenie rysunków

Aby umieścić wiele wykresów funkcji na jednym rysunku, możemy łączyć poszczególne obiekty graficzne za pomocą operatora +:

```
plot(x^2, color='green') + plot(-x^2 + 1, color='orange')
```

W przypadku bardziej skomplikowanych rysunków taki zapis staje się jednak nieczytelny. W związku z tym warto przypisywać kolejne obiekty graficzne do zmiennych:

```
rys1 = plot(x^2, color='green')
rys2 = plot(-x^2 + 1, color='orange')
rys1 + rys2
```

Możemy również użyć operatora +=, aby do istniejącego rysunku dodawać kolejne wykresy:

```
rys = plot(x^2, color='green')
rys += plot(-x^2 + 1, color='orange')
rys
```

Alternatywnie dla funkcji **plot()** ten sam efekt można osiągnąć poprzez przekazanie wzorów funkcji w liście oraz odpowiadających im kolorów:

```
plot( [x^2, -x^2+1], color=['green', 'orange'] )
```

Pusty rysunek tworzymy za pomocą komendy Graphics(). Następnie możemy go rozbudowywać, dodając do niego kolejne elementy za pomocą operatora +=:

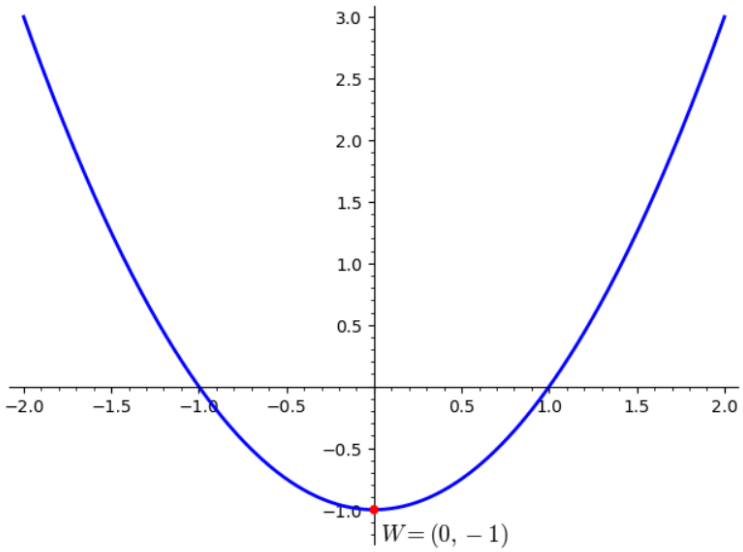
```
rys = Graphics()
rys += plot( x^2 - 1, (x, -2, 2), thickness=2 )
```

[7] Sage, *2D plotting*, https://doc.sagemath.org/html/en/reference/plotting/sage/plot/plot.html [dostęp: 18.10.2025].

```
rys += point( (0, -1), color='red', size=30, zorder=5 )
rys
```

Aby umieścić tekst na rysunku, korzystamy z funkcji **text()**:

```
rys += text( '$W=(0, -1)$', (0.4, -1.2), color='black', fontsize=14 )
rys
```



Rysunek 2.13. Wykres funkcji kwadratowej z zaznaczonym wierzchołkiem oraz tekstem na rysunku

Efekt końcowy działania powyższego kodu przedstawiony został na rysunku 2.13. Pierwszy argument funkcji **text()** jest tekstem wyświetlanym na rysunku. Drugi argument określa współrzędne, na których tekst zostanie umieszczony. Opcjonalne argumenty color i fontsize modyfikują natomiast odpowiednio kolor tekstu oraz rozmiar czcionki. Aby zapisać końcowy rysunek do pliku w formacie **.pdf**, korzystamy z polecenia:

```
rys.save('nazwa_pliku.pdf')
```

Uwaga. W notatniku **Jupyter** obiekt graficzny jest automatycznie wyświetlany, jeśli znajduje się w ostatniej linii komórki. Aby pokazać rysunek w dowolnym miejscu lub wyświetlić wiele wykresów jednocześnie, należy użyć funkcji **show()**.

2.4. Równania i układy równań

SageMath oferuje narzędzia do pracy zarówno z pojedynczymi równaniami, jak i układami równań. Możemy rozwiązywać nie tylko równania liniowe i nieliniowe, ale także równania zawierające parametry lub wiele niewiadomych.

Równania w **SageMath** można zapisywać w formie symbolicznej – wówczas system podejmie próbę znalezienia rozwiązania dokładnego za pomocą odpowiednich algorytmów. W sytuacji, gdy uzyskanie analitycznego rozwiązania jest niemożliwe, możemy wykorzystać metody numeryczne, aby znaleźć wynik przybliżony.

2.4.1. Rozwiązywanie równań i nierówności

Podstawowym narzędziem do rozwiązywania równań i nierówności w **Sage** jest funkcja **solve()**. Pierwszym argumentem tej funkcji jest równość, którą zapisujemy za pomocą operatora ==. Drugi argument określa zmienną, względem której szukamy rozwiązania.

Przykład 2.4
Rozwiążemy równanie:

$$2x^2 - 5x = 3.$$

Stosujemy składnię **solve**(*równość*, *zmienna*). W wyniku otrzymujemy listę równości zawierających rozwiązania równania:

```
solve(2*x^2 - 5*x == 3, x)
Out: [x == 3, x == (-1/2)]
```

Jeżeli po prawej stronie równania znajduje się liczba 0, możemy pominąć znak równości. Funkcja **solve()** zakłada domyślnie, że prawa strona jest równa 0, dlatego ten sam przykład można zapisać krócej:

```
solve(2*x^2 - 5*x - 3, x)
Out: [x == 3, x == (-1/2)]
```

Przykład 2.5
Rozwiążemy nierówność:

$$x^2 - 2x - 3 > 0.$$

Nierówności rozwiązujemy podobnie jak równania – za pomocą funkcji **solve()**:

```
solve(x^2 - 2*x - 3 > 0, x)
Out: [[x < -1], [x > 3]]
```

Funkcja zwraca listę zawierającą rozwiązania nierówności. W **Sage** nawiasy kwadratowe oddzielone przecinkiem, na przykład `[[x < -1], [x > 3]]`, oznaczają sumę przedziałów, czyli zbiór $(-\infty, -1) \cup (3, +\infty)$. Zapis `[[x > -1, x < 3]]` rozumiemy zaś jako część wspólną przedziałów, czyli zbiór $(-1, 3)$.

Przykład 2.6
Rozwiążemy poniższe równanie oraz podamy przybliżoną wartość pierwszego rozwiązania:

$$x^3 + 2x^2 = 3x + 6.$$

Najpierw wypisujemy wszystkie rozwiązania równania:

```
rozw = solve(x^3 + 2*x^2 == 3*x + 6, x)
rozw
Out: [x == -sqrt(3), x == sqrt(3), x == -2]
```

Wynik zwracany przez funkcję **solve()** przypisujemy do zmiennej rozw. Ponieważ jest to lista, możemy odwoływać się do poszczególnych rozwiązań za pomocą indeksów. W języku **Python** obowiązuje indeksowanie od 0, więc pierwszy element listy rozw uzyskujemy za pomocą komendy `rozw[0]`, drugi jako `rozw[1]` itd. Ostatni element listy możemy wyodrębnić, korzystając z polecenia `rozw[-1]` – ujemne indeksy w **Pythonie** oznaczają odliczanie od końca. Wypisujemy pierwsze rozwiązanie równania:

```
rozw[0]
Out: x == -sqrt(3)
```

Jest to równość, zatem nie możemy bezpośrednio obliczyć jej wartości przybliżonej, ponieważ otrzymalibyśmy błąd. Najpierw należy wyodrębnić prawą stronę tej równości (**Right-Hand Side**) za pomocą funkcji **rhs()**:

```
rozw[0].rhs()
Out: -sqrt(3)
```

Aby podać wartość przybliżoną tego rozwiązania, korzystamy z funkcji **n()**:

```
rozw[0].rhs().n()
Out: -1.73205080756888
```

Przykład 2.7

Rozwiążemy równanie

$$x^4 - 1 = 0$$

i wypiszemy w pętli wszystkie jego rzeczywiste rozwiązania.

Pętla to konstrukcja umożliwiająca wielokrotne wykonywanie powtarzalnych operacji. W języku **Python** najczęściej stosujemy pętlę **for**, która pozwala iterować (przechodzić) po elementach listy. Poniżej iterujemy w pętli **for** po elementach listy przedmioty i wypisujemy jej zawartość na ekran:

```
przedmioty = ['algebra', 'analiza', 'programowanie']
for przedmiot in przedmioty:
    print(przedmiot)
Out: algebra
 analiza
 programowanie
```

W pierwszej iteracji zmienna przedmiot przyjmuje wartość 'algebra', w drugiej 'analiza' itd. W języku **Python** po symbolu dwukropka (:) należy stosować wcięcie za pomocą tabulatora.

Wykorzystujemy tę składnię, aby wypisać rozwiązania równania w przykładzie:

```
for r in solve(x^4 - 1, x):
    print( r.rhs() )
```

```
Out: I
 -1
 -I
 1
```

Funkcja **solve()** zwraca listę `[x == I, x == -1, x == -I, x == 1]`. W pętli iterujemy po elementach tej listy, więc do zmiennej `r` w kolejnych iteracjach przypisywane są poszczególne równości zawarte w liście. Wewnątrz pętli **for** wypisujemy natomiast na ekran prawe strony tych równości, korzystając z funkcji **rhs()**. Niektóre rozwiązania tego równania są liczbami zespolonymi, ponieważ zawierają jednostkę urojoną `I`. Więcej na temat takich liczb powiemy w następnym rozdziale – na razie wystarczy nam wiedza, że nie są to liczby rzeczywiste.

Aby określić założenia dotyczące dziedziny równania, korzystamy z funkcji **assume()**. W poniższym kodzie zakładamy, że dziedziną jest zbiór liczb rzeczywistych:

```
assume(x, 'real')
for r in solve(x^4 - 1, x):
   print( r.rhs() )
Out: -1
 1
```

Aktualne założenia możemy wyświetlić za pomocą funkcji **assumptions()**, natomiast aby zapomnieć o wszystkich założeniach, stosujemy polecenie:

```
forget()
```

Zadanie można również rozwiązać za pomocą instrukcji warunkowej **if**. Jest to konstrukcja, która pozwala na wykonanie określonego bloku kodu w zależności od tego, czy spełniony jest warunek logiczny. Poniższy przykład ilustruje jej działanie:

```
wiek = 19
if wiek < 18:
   print('Nie masz 18 lat!')
else:
   print('Masz ukończone 18 lat!')
Out: Masz ukończone 18 lat!
```

Jeżeli warunek `wiek < 18` jest spełniony, wykonuje się pierwszy blok kodu po dwukropku. W przeciwnym razie przechodzimy do drugiego bloku, znajdującego się po instrukcji **else**.

W języku **Sage** funkcjonuje następujące nazewnictwo zbiorów liczbowych:

- `NN` – zbiór liczb naturalnych (z zerem);
- `ZZ` – zbiór liczb całkowitych;
- `QQ` – zbiór liczb wymiernych;
- `RR` – zbiór liczb rzeczywistych;
- `CC` – zbiór liczb zespolonych.

Poniżej przedstawiono rozwiązanie przykładu 2.7, które wykorzystuje instrukcję warunkową **if**:

```
for r in solve(x^4 - 1, x):
    if r.rhs() in RR:
        print(r)
Out: -1
 1
```

Uwaga. Wyrażenie `r.rhs() in RR` w powyższej pętli sprawdza, czy prawa strona równości jest liczbą rzeczywistą, natomiast zapis `r in RR` nigdy nie będzie prawdziwy, ponieważ `r` jest równością, a nie liczbą.

2.4.2. Numeryczne rozwiązywanie równań

Gdy równanie jest zbyt skomplikowane, możemy znaleźć jego rozwiązanie numerycznie w określonym przedziale, korzystając z funkcji **find_root()**.

Przykład 2.8
Rozwiążemy numerycznie równanie:

$$e^x + \ln x = x.$$

Funkcja **solve()** nie zwraca żadnego wyniku, ponieważ to równanie jest zbyt trudne. Otrzymujemy jedynie nieco przekształconą wersję pierwotnej równości:

```
rownanie = exp(x) + log(x) == x
solve(rownanie, x)
Out: [x == e^x + log(x)]
```

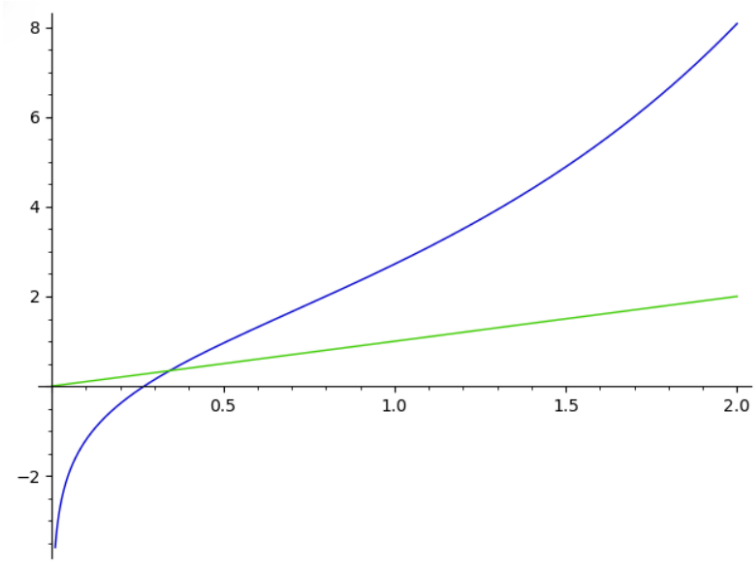
W takiej sytuacji należy rozwiązać równanie numerycznie, używając składni:

find_root(*równość*, *lewy*, *prawy*).

Wówczas funkcja automatycznie zastosuje odpowiednią metodę obliczeniową i zwróci jedno rozwiązanie (lub informację o jego braku) w przedziale [*lewy*, *prawy*]. Jeżeli w zadanym przedziale jest więcej pierwiastków, trzeba odpowiednio zawęzić przeszukiwany przedział, tak aby znaleźć wszystkie rozwiązania:

```
find_root(rownanie, 0, 1)
Out: 0.34416128672195007
```

W wyniku pojawia się również komunikat o przedawnieniu (**Deprecation**) funkcji **find_root()**. Oznacza to, że w przyszłych wersjach **SageMath** funkcja ta może zostać usunięta lub zastąpiona innym narzędziem. Aby ukryć to ostrzeżenie, wystarczy ponownie uruchomić kod w tej samej komórce.



Rysunek 2.14. Rysunek pomocniczy – pierwsza współrzędna punktu przecięcia wykresów funkcji jest szukanym rozwiązaniem równania

Skąd wiadomo, że rozwiązania należy szukać akurat w przedziale $[0, 1]$? Najpierw warto narysować wykresy funkcji występujących po obu stronach równania, aby zorientować się, gdzie znajduje się potencjalne rozwiązanie:

```
plot( [rownanie.lhs(), rownanie.rhs()], (x, 0, 2) )
```

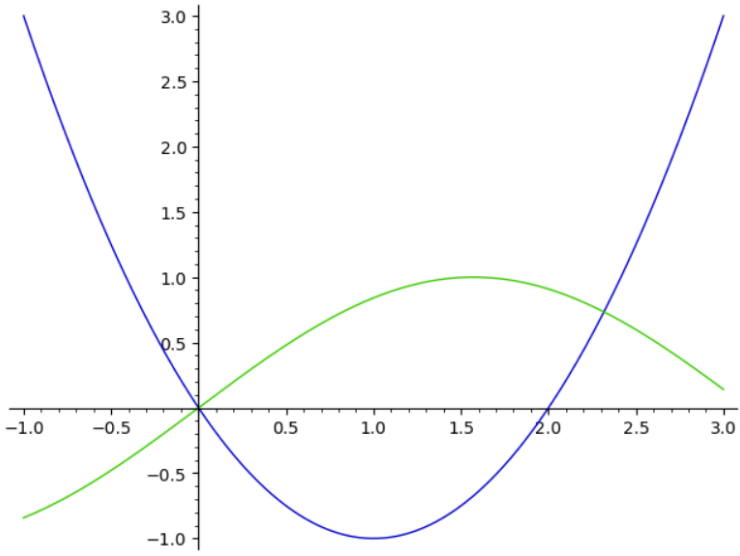
Wynik działania tego kodu przedstawiono na rysunku 2.14. Umieszczamy w liście lewą i prawą stronę równania, co sprawia, że obie funkcje rysowane są różnymi kolorami na jednym rysunku. Widzimy, że wykresy przecinają się w punkcie o pierwszej współrzędnej $x \approx 0{,}35$, więc spodziewamy się rozwiązania w przedziale $[0, 1]$. Na tej podstawie wybieramy wartości przekazywane do funkcji **find_root()**.

Przykład 2.9
Rozwiążemy numerycznie równanie:

$$x^2 - 2x = \sin x.$$

Zaczynamy od rysunku pomocniczego, aby zlokalizować potencjalne rozwiązania:

```
rownanie = x^2 - 2*x == sin(x)
plot( [rownanie.lhs(), rownanie.rhs()], (x, -1, 3) )
```



Rysunek 2.15. Rysunek pomocniczy do znalezienia rozwiązań równania $x^2 - 2x = \sin x$

Efekt działania tego kodu przedstawiono na rysunku 2.15. Tym razem dostrzegamy dwa punkty przecięcia. Znajdują się one na przykład w przedziałach $[-1, 1]$ oraz $[2, 3]$. W związku z tym rozwiązania tego równania znajdujemy następująco:

```
print( 'Pierwsze rozwiązanie:', find_root(rownanie, -1, 1) )
print( 'Drugie rozwiązanie:', find_root(rownanie, 2, 3) )
Out: Pierwsze rozwiązanie: 2.0158492568029108e-16
 Drugie rozwiązanie: 2.3169342886238704
```

Pierwsza wartość jest liczbą równą około $2 \cdot 10^{-16}$, więc jest to w przybliżeniu 0.

Ponownie – więcej informacji dotyczących rozwiązywania równań można znaleźć w dokumentacji równań i nierówności[8] w **Sage**.

2.4.3. Rozwiązywanie układów równań

Układy równań również można rozwiązywać za pomocą funkcji **solve()**. Pierwszym argumentem jest lista równości występujących w układzie, natomiast drugim – lista zmiennych, względem których chcemy go rozwiązać.

Przykład 2.10
Rozwiążemy układ równań:

$$\begin{cases} -x + 2y - z = 6 \\ 2x + y + 2z = 3 \\ 3x - y + z = -2. \end{cases}$$

W układzie występują trzy zmienne: x, y i z. Najpierw należy zadeklarować je jako symbole za pomocą funkcji **var()**. Następnie umieszczamy poszczególne równości w liście będącej pierwszym argumentem funkcji **solve()**:

```
var('x y z')
solve( [-x+2*y-z==6, 2*x+y+2*z==3, 3*x-y+z==-2], [x, y, z] )
Out: [[x == (1/2), y == 3, z == (-1/2)]]
```

[8] Sage, *Symbolic equations and inequalities*, https://doc.sagemath.org/html/en/reference/calculus/sage/symbolic/relation.html [dostęp: 18.10.2025].

Przykład 2.11

Rozwiążemy układ równań liniowych, który ma nieskończenie wiele rozwiązań:

$$\begin{cases} x + 2y + z + t = 1 \\ x + 2y + 5z + 3t = 3 \\ 2x + 4y - z - t = 2. \end{cases}$$

Parametry w **Sage** reprezentowane są przez symbole `r1`, `r2`, `r3` itd., które są generowane automatycznie przy kolejnych uruchomieniach komórek w notatniku. W poniższym kodzie rozwiązania zależne są od jednego parametru `r1`:

```
var('x y z t')
solve( [x+2*y+z+t==1,
 x+2*y+5*z+3*t==3,
 2*x+4*y-z-t==2], [x, y, z, t] )
Out: [[x == -2*r1 + 1, y == r1, z == 1, t == -1]]
```

Przykład 2.12

Rozwiążemy układ równań nieliniowych:

$$\begin{cases} 3y^2 - x^2 + x = 2 \\ 5x^2 + y + 4 = 0. \end{cases}$$

Ponownie, aby uzyskać rozwiązania układu, korzystamy z funkcji **solve()**. Warto zwrócić uwagę, że poszczególne równości można przypisywać do zmiennych:

```
r1 = 3*y^2 - x^2 + x == 2
r2 = 5*x^2 + y + 4 == 0
rozw = solve( [r1, r2], [x, y] )
rozw
Out: [[x == (-0.02484220293816692 + 0.8122623991014778*I),
 y == (-0.7042346502636634 + 0.20178387357521244*I)],
 [x == (-0.02484220293816692 - 0.8122623991014778*I),
 y == (-0.7042346502636634 - 0.20178387357521244*I)],
 [x == (0.02484220293816692 + 0.9633954181779256*I),
```

```
y == (0.6375679835969956 - 0.2393286448807621 4*I)],
[x == (0.02484220293816692 - 0.9633954181779256*I),
y == (0.6375679835969956 + 0.2393286448807621 4*I)]]
```

Otrzymujemy listę zawierającą cztery rozwiązania zespolone – w każdym z nich występuje symbol I oznaczający jednostkę urojoną. Na podstawie tych wyników wnioskujemy, że układ ma tylko rozwiązania zespolone – nie ma rozwiązań w zbiorze liczb rzeczywistych.

Podobnie jak w przypadku pojedynczych równań, do rozwiązań można odwoływać się za pomocą indeksów. Aby wyodrębnić ostatnie rozwiązanie układu, piszemy:

```
rozw[-1]
Out: [x == (0.02484220293816692 - 0.9633954181779256*I),
y == (0.6375679835969956 + 0.2393286448807621 4*I)]
```

Uzyskany wynik jest listą. Możemy teraz stosować podwójne indeksowanie, aby uzyskać dostęp do elementów znajdujących się na kolejnym poziomie zagnieżdżenia. Wypisujemy pierwszą zmienną w ostatnim rozwiązaniu układu:

```
rozw[-1][0]
Out: x == (0.02484220293816692 - 0.9633954181779256*I)
```

Poniższy kod zwraca wartość pierwszej zmiennej w ostatnim rozwiązaniu:

```
rozw[-1][0].rhs()
Out: 0.02484220293816692 - 0.9633954181779256*I
```

Więcej informacji na temat możliwości języka **Sage** oraz praktycznych przykładów jego zastosowania w obliczeniach matematycznych można znaleźć w pozycji [1].

Miłego dnia! 😀

3. Liczby zespolone

3.1. Ciało liczb zespolonych

Zbiór liczb zespolonych $\mathbb{C}$ stanowi uogólnienie zbioru liczb rzeczywistych $\mathbb{R}$, w którym wprowadzamy jednostkę urojoną i, czyli liczbę o własności:

$$i^2 = -1.$$

Oznacza to, że kwadrat liczby zespolonej nie musi być nieujemny. Ponadto, w przeciwieństwie do liczb rzeczywistych, liczb zespolonych nie da się ze sobą porównywać.

Liczby zespolone można utożsamiać z punktami na płaszczyźnie. Przykładowo liczbę zespoloną $3 + 2i$ identyfikujemy z punktem $(3, 2)$ w układzie współrzędnych. Uzyskaną w ten sposób konstrukcję nazywamy płaszczyzną zespoloną. Wyróżniamy na niej:

- poziomą oś rzeczywistą, na której znajdują się liczby rzeczywiste;
- pionową oś urojoną składającą się z tzw. liczb czysto urojonych, na przykład $-4i$.

Zbiór liczb zespolonych wraz ze zwykłymi działaniami dodawania i mnożenia stanowi tzw. ciało. Jest to struktura zamknięta ze względu na te działania (tzn. wynik zawsze należy do początkowego zbioru), w której obowiązują standardowe reguły działań.

W ciele liczb zespolonych każde równanie wielomianowe n-tego stopnia zawsze ma dokładnie n rozwiązań (licząc z krotnościami). Na przykład równanie kwadratowe

$$x^2 + 1 = 0$$

ma dokładnie dwa rozwiązania zespolone, mimo że wyróżnik Δ przyjmuje wartość ujemną. Są to liczby należące do zbioru $\{-i, i\}$.

3.1.1. Postać algebraiczna liczby zespolonej

Definicja. Postacią algebraiczną liczby zespolonej $z \in \mathbb{C}$ nazywamy zapis:

$$z = x + yi, \quad \text{gdzie } x, y \in \mathbb{R}. \tag{3.1}$$

Wyróżniamy tutaj:

- część rzeczywistą (**Real Part**) $\operatorname{Re} z := x \in \mathbb{R}$;
- część urojoną (**Imaginary Part**) $\operatorname{Im} z := y \in \mathbb{R}$.

Uwaga. Część rzeczywista i część urojona są liczbami rzeczywistymi.

Przykład 3.1
Znajdziemy część rzeczywistą i część urojoną liczby zespolonej:

$$z = 5 - 2i.$$

Na podstawie definicji odczytujemy części rzeczywistą i urojoną liczby zespolonej z jako:

$$\operatorname{Re} z = 5 \quad \text{oraz} \quad \operatorname{Im} z = -2.$$

W **Sage** części rzeczywistą i urojoną uzyskujemy odpowiednio za pomocą funkcji **real()** oraz **imag()**:

```
z = 5 - 2*i
print( 'Część rzeczywista:', real(z) )
print( 'Część urojona:', imag(z) )
Out: Część rzeczywista: 5
 Część urojona: -2
```

Potęgi jednostki urojonej i powtarzają się cyklicznie co 4, co zostało przedstawione w tabeli 3.1.

Tabela 3.1. Cykliczność potęg jednostki urojonej

$i^1 = i$	$i^5 = i$	$i^9 = i$
$i^2 = -1$	$i^6 = -1$	$i^{10} = -1$
$i^3 = -i$	$i^7 = -i$	$i^{11} = -i$
$i^4 = 1$	$i^8 = 1$	$i^{12} = 1$

Przykład 3.2
Obliczymy potęgę:

$$i^{111}.$$

Obserwacje zebrane w tabeli 3.1 pozwalają szybko liczyć wysokie potęgi jednostki urojonej. Najpierw znajdujemy liczbę podzielną przez 4 w pobliżu wykładnika – dla 111 jest to 108. Następnie możemy obniżyć wykładnik potęgi:

$$i^{111} = i^{108+3} = i^3 = -i.$$

W języku **Sage** jednostka urojona reprezentowana jest za pomocą wielkiej lub małej litery i. System domyślnie zapisuje wynik w najprostszej postaci:

```
i^111
Out: -I
```

Ponieważ liczbę zespoloną

$$z = x + yi$$

utożsamiamy z punktem na płaszczyźnie o współrzędnych:

$$(x, y),$$

zbiór liczb zespolonych $\mathbb{C}$ można traktować jako zbiór par uporządkowanych złożonych z liczb rzeczywistych, czyli jest to płaszczyzna:

$$\mathbb{C} = \mathbb{R} \times \mathbb{R} = \mathbb{R}^2.$$

Przykład 3.3
Na płaszczyźnie zespolonej zaznaczymy punkty odpowiadające liczbom:

$$z = i \quad \text{oraz} \quad \omega = -1 - i.$$

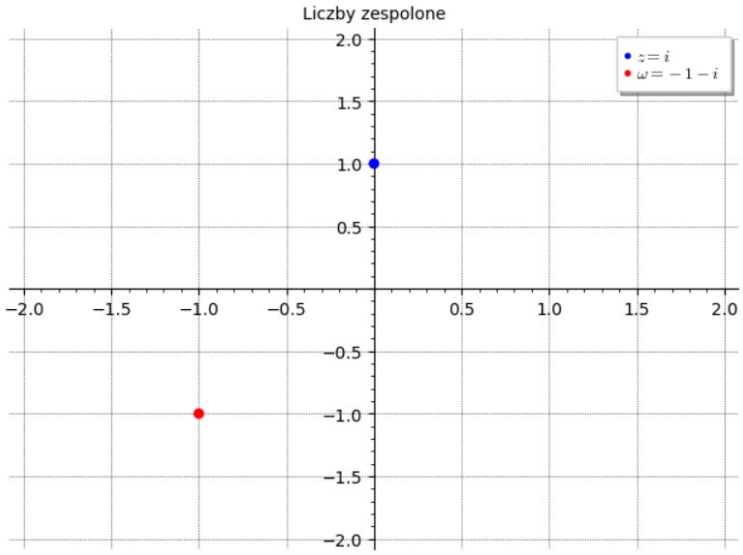
Aby zaznaczyć liczby zespolone na rysunku, korzystamy z funkcji **point()** lub **points()** – tej drugiej używamy, gdy chcemy umieścić wiele punktów jednocześnie:

```
z = i
w = -1 - i

rys = point( z, size=40, zorder=5, gridlines=True,
 title='Liczby zespolone', legend_label='$z=i$' )
rys += point( w, color='red', size=40, zorder=5,
 legend_label=r'$\omega=-1-i$' )
```

```
rys.axes_range(-2, 2, -2, 2)
rys
```

Wynik przedstawiono na rysunku 3.1. Zadane liczby zespolone przypisujemy do zmiennych z i w. Za pomocą pierwszej funkcji **point()** umieszczamy na rysunku niebieski punkt, tytuł, legendę oraz linie siatki. Druga natomiast dodaje czerwony punkt przy użyciu operatora +=. Funkcja **axes_range()** określa zakres wyświetlanych wartości. Pierwsze dwie liczby definiują zakres na osi poziomej, a dwie kolejne – na osi pionowej. Wobec tego:

- na osi OX znajdują się zwykłe liczby rzeczywiste $x \in \mathbb{R}$ – odpowiadają im punkty postaci $(x, 0)$;
- na osi OY położone są liczby czysto urojone $yi \in \mathbb{C}$, którym odpowiadają punkty postaci $(0, y)$.



Rysunek 3.1. Liczby zespolone i oraz $-1 - i$ na płaszczyźnie zespolonej

Definicja. Sprzężeniem (**Conjugate**) liczby zespolonej $z = x + yi$ nazywamy liczbę zespoloną postaci:

$$\overline{z} = x - yi. \tag{3.2}$$

Przykład 3.4
Znajdziemy sprzężenie liczby:

$$z = 2 - 3i.$$

Sprzężenie $\overline{z}$ zmienia znak części urojonej liczby zespolonej z, więc na płaszczyźnie zespolonej punkty odpowiadające liczbie i jej sprzężeniu są symetryczne względem osi rzeczywistej. Otrzymujemy:

$$\overline{z} = 2 + 3i.$$

W **Sage** sprzężenie liczby zespolonej obliczamy za pomocą funkcji **conjugate()**. W poniższym kodzie wykorzystano tzw. **f-string** z języka **Python**, który charakteryzuje się literą f przed tekstem zawartym między apostrofami. Składnia ta pozwala wstawić w tekście wartość zmiennej umieszczonej w nawiasach klamrowych. Co więcej, zapis {z = } powoduje wyświetlenie nazwy zmiennej wraz z jej zawartością:

```
z = 2 - 3*i
print(f'Sprzężenie liczby zespolonej {z = } to:')
conjugate(z)
Out: Sprzężenie liczby zespolonej z = -3*I + 2 to:
 3*I + 2
```

Definicja. Moduł (**Absolute Value**) liczby zespolonej $z = x + yi$ jest liczbą rzeczywistą, którą obliczamy ze wzoru:

$$|z| = \sqrt{x^2 + y^2}. \tag{3.3}$$

Na płaszczyźnie zespolonej jest to długość tzw. promienia wodzącego, czyli wektora łączącego początek układu współrzędnych z punktem o współrzędnych (x, y).

Przykład 3.5
Obliczymy moduł poniższej liczby oraz przedstawimy jego interpretację na rysunku:

$$z = -3 + 4i.$$

Liczymy moduł liczby zespolonej z, korzystając ze wzoru (3.3):

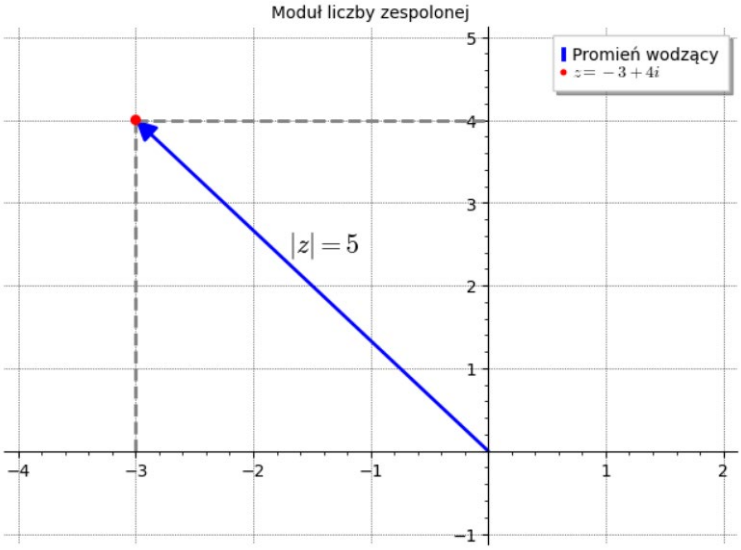
$$|z| = \sqrt{(-3)^2 + 4^2} = \sqrt{25} = 5.$$

W **Sage** wykonujemy to działanie za pomocą funkcji **abs()**:

```
z = -3 + 4*i
print( 'Moduł liczby zespolonej:', abs(z) )
Out: Moduł liczby zespolonej: 5
```

Następnie zaznaczamy promień wodzący na rysunku. Moduł jest długością odcinka łączącego początek układu współrzędnych z punktem $(-3, 4)$:

```
rys = plot( vector(z), gridlines=True,
 title='Moduł liczby zespolonej',
 legend_label='Promień wodzący' )
rys += point( z, color='red', size=40, zorder=5,
 legend_label='$z=-3+4i$' )
rys += text( '$|z|=5$', (-1.4, 2.5),
 color='black', fontsize=16 )
rys += line( [(-3, 0), (-3, 4)], color='gray',
 thickness=2, linestyle='--' )
rys += line( [(-3, 4), (0, 4)], color='gray',
 thickness=2, linestyle='--' )

rys.axes_range(-4, 2, -1, 5)
rys
```



Rysunek 3.2. Interpretacja geometryczna modułu liczby zespolonej $z = -3 + 4i$

Efekt działania kodu przedstawiono na rysunku 3.2. Funkcja **plot()** generuje niebieską strzałkę reprezentującą promień wodzący przy użyciu funkcji **vector()**. Liczbę zespoloną z dodajemy do rysunku za pomocą funkcji **point()**. Następnie umieszczamy na nim tekst oraz linie pomocnicze zaznaczone stylem przerywanym.

Definicja. Argumentem (**Argument**) liczby zespolonej z nazywamy miarę kąta skierowanego φ (wyrażonego w radianach) pomiędzy dodatnią półosią osi rzeczywistej a promieniem wodzącym liczby z. Oznaczamy go symbolem:

$$\arg z := \varphi.$$

Argument liczby zespolonej nie jest określony jednoznacznie, ponieważ jego wartości mogą różnić się o dowolną wielokrotność kąta 2π. Dlatego wprowadzamy pojęcie argumentu głównego, który oznaczamy jako:

$$\mathrm{Arg}\, z.$$

W praktyce stosowane są dwie konwencje określania argumentu głównego:
- liczba z przedziału $[0, 2\pi)$;
- liczba z przedziału $(-\pi, \pi]$.

W **Sage** obowiązuje druga umowa – argument główny należy do przedziału $(-\pi, \pi]$.

Przykład 3.6
Znajdziemy argument główny liczby zespolonej

$$z = \sqrt{3} + i$$

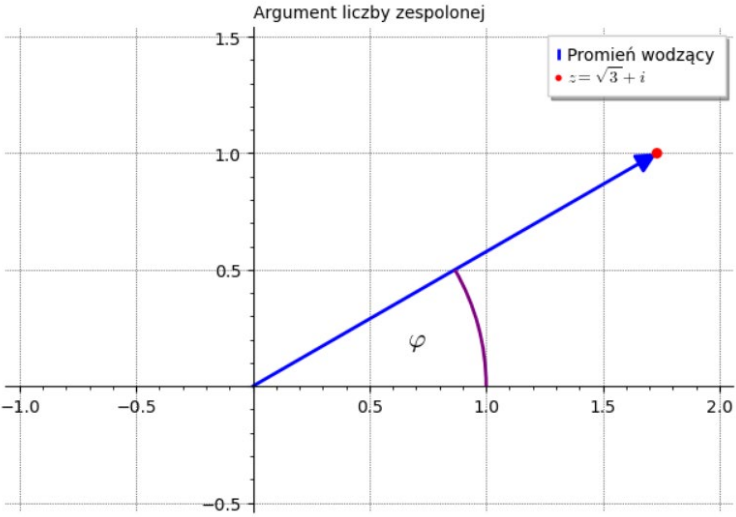
oraz przedstawimy jego interpretację graficzną.

Argument obliczamy za pomocą funkcji **arg()**. W tym przykładzie wartość ta nie jest domyślnie zapisywana w najprostszej postaci, dlatego stosujemy funkcję **simplify()**:

```
z = sqrt(3) + i
phi = arg(z).simplify()
print('Argument główny liczby zespolonej:', phi)
Out: Argument główny liczby zespolonej: 1/6*pi
```

Interpretacją graficzną argumentu jest kąt między dodatnią półosią poziomej osi rzeczywistej a promieniem wodzącym liczby zespolonej z:

```
rys = plot( vector(z.n()), gridlines=True,
 title='Argument liczby zespolonej',
 legend_label='Promień wodzący' )
rys += point( z, color='red', size=40, zorder=5,
 legend_label=r'$z=\sqrt{3}+i$' )
rys += arc( (0, 0), 1, sector=(0, phi),
 thickness=2, color='purple' )
rys += text( r'$\varphi$', (0.7, 0.2),
 color='black', fontsize=16 )

rys.axes_range(-1, 2, -0.5, 1.5)
rys
```



Rysunek 3.3. Interpretacja graficzna argumentu głównego liczby zespolonej $z = \sqrt{3} + i$

Uzyskany wynik przedstawiony został na rysunku 3.3. Tym razem do funkcji **vector()** przekazujemy przybliżenie numeryczne liczby $z = \sqrt{3} + i$ – należy tak robić, gdy zmienna zawiera pierwiastki (w przeciwnym razie otrzymalibyśmy błąd). Funkcja **arc()** rysuje łuk o środku w punkcie $(0, 0)$ i promieniu 1 dla rozpiętości zadanej przez argument sector. Następnie dodajemy do rysunku tekst oraz wyświetlamy go w wybranym miejscu.

3.1.2. Działania na liczbach zespolonych

Przeanalizujemy podstawowe działania w zbiorze liczb zespolonych $\mathbb{C}$ oraz ich interpretację geometryczną:

1. Dodawanie i odejmowanie.
Dla dwóch liczb zespolonych, $z_1 = x_1 + y_1 i$ oraz $z_2 = x_2 + y_2 i$, definiujemy działania:

$$z_1 \pm z_2 = (x_1 \pm x_2) + (y_1 \pm y_2)i.$$

Przykład 3.7
Znajdziemy sumę liczb zespolonych:

$$z_1 = -2 + 3i \quad \text{oraz} \quad z_2 = 3 - i.$$

Dodajemy oddzielnie części rzeczywiste i urojone:

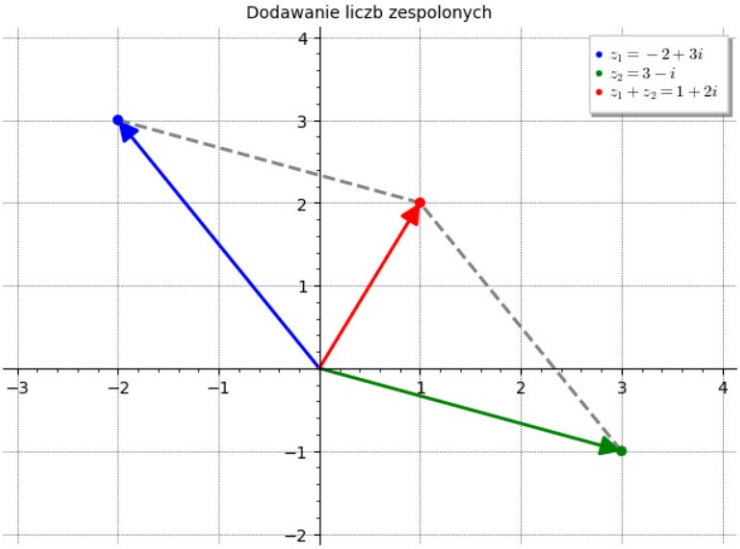
$$z_1 + z_2 = (-2 + 3i) + (3 - i) = (-2 + 3) + (3i - i) = 1 + 2i.$$

W **Sage** korzystamy z operatora dodawania +:

```
z1 = -2 + 3*i
z2 = 3 - i
print('Suma liczb zespolonych:', z1+z2)
Out: Suma liczb zespolonych: 2*I + 1
```

Na płaszczyźnie zespolonej działanie to wykonujemy tak samo jak dodawanie wektorów – stosujemy regułę równoległoboku:

```
rys = plot( vector(z1), gridlines=True,
 title='Dodawanie liczb zespolonych' )
rys += plot( vector(z2), color='green' )
rys += plot( vector(z1+z2), color='red' )
rys += point( z1, size=40, zorder=5,
 legend_label='$z_1=-2+3i$')
rys += point( z2, color='green', size=40, zorder=5,
 legend_label='$z_2=3-i$' )
rys += point( z1+z2, color='red', size=40, zorder=5,
 legend_label='$z_1+z_2=1+2i$' )
rys += line( [(-2, 3), (1, 2)], color='gray',
 thickness=2, linestyle='--' )
rys += line( [(3, -1), (1, 2)], color='gray',
 thickness=2, linestyle='--' )

rys.axes_range(-3, 4, -2, 4)
rys
```



Rysunek 3.4. Interpretacja geometryczna dodawania liczb zespolonych

Wynik przedstawiono na rysunku 3.4. Wektory wodzące poszczególnych liczb zespolonych zostały narysowane za pomocą funkcji **plot()** – sumę zaznaczono kolorem czerwonym. Funkcje **point()** dodają do rysunku punkty reprezentujące liczby zespolone, natomiast **line()** – linie pomocnicze tworzące równoległobok. Odejmowanie liczb zespolonych wykonujemy analogicznie – dla danej liczby $z = x + yi$, gdzie $x, y \in \mathbb{R}$, liczba przeciwna, $-z = -x - yi$, jest reprezentowana przez wektor o tej samej długości, ale przeciwnym zwrocie w porównaniu do wektora odpowiadającego liczbie z.

2. Mnożenie.
Dla dwóch liczb zespolonych, $z_1 = x_1 + y_1 i$ oraz $z_2 = x_2 + y_2 i$, mnożenie określamy w naturalny sposób jako:

$$z_1 \cdot z_2 = (x_1 x_2 - y_1 y_2) + (x_1 y_2 + x_2 y_1) i.$$

Przykład 3.8
Obliczymy iloczyn liczb zespolonych:

$$z_1 = 1 + 3i \quad \text{oraz} \quad z_2 = i.$$

Wykonujemy rachunki, pamiętając o zależności $i^2 = -1$. Uzyskujemy:

$$z_1 \cdot z_2 = (1 + 3i) \cdot i = 1 \cdot i + 3i \cdot i = i + 3 \cdot (-1) = -3 + i.$$

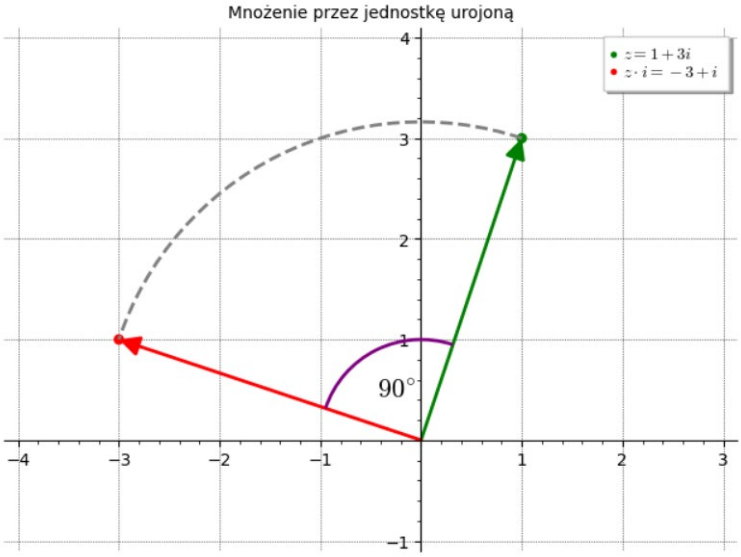
Aby zapisać to działanie w **Sage**, korzystamy z operatora mnożenia *:

```
z1 = 1 + 3*i
z2 = i
print('Iloczyn liczb zespolonych:', z1*z2)
Out: Iloczyn liczb zespolonych: I – 3
```

Na płaszczyźnie zespolonej mnożenie przez jednostkę urojoną *i* odpowiada obrotowi danej liczby o kąt 90° przeciwnie do ruchu wskazówek zegara wokół początku układu współrzędnych:

```
rys = plot( vector(z1), color='green', gridlines=True,
 title='Mnożenie przez jednostkę urojoną' )
rys += plot( vector(z1*z2), color='red' )
rys += point( z1, color='green', size=40, zorder=5,
 legend_label='$z=1+3i$' )
rys += point( z1*z2, color='red', size=40, zorder=5,
 legend_label=r'$z\cdot i=-3+i$' )
rys += arc( (0, 0), 1, sector=(arg(z1), arg(z1*z2)),
 thickness=2, color='purple' )
rys += arc( (0, 0), abs(z1), sector=(arg(z1), arg(z1*z2)),
 thickness=2, color='gray', linestyle='--' )
rys += text( r'$90^\circ$', (-0.25, 0.5),
 color='black', fontsize=16 )

rys.axes_range(-4, 3, -1, 4)
rys
```

Efekt działania kodu został przedstawiony na rysunku 3.5. Funkcje **arc()** dodają łuki: wewnętrzny (fioletowy) o promieniu 1 i zewnętrzny (zaznaczony linią przerywaną) o promieniu równym modułowi liczby z_1 (jest to długość strzałki na rysunku). Funkcja **text()** umieszcza napis 90° w miejscu określonym przez wprowadzone współrzędne. Interpretacja ta wynika z własności mnożenia liczb zespolonych zapisanych w postaci wykładniczej, o której będzie mowa w dalszej części książki.



Rysunek 3.5. Interpretacja graficzna mnożenia liczby zespolonej przez jednostkę urojoną

3. Dzielenie.
W celu wyznaczenia ilorazu dwóch liczb zespolonych rozszerzamy ułamek przez sprzężenie dzielnika na podstawie wzoru (3.2). Przykład 3.9 pokazuje ten sposób obliczeń.

Przykład 3.9
Wyznaczymy iloraz liczb zespolonych:

$$z_1 = 1 + 2i \quad \text{oraz} \quad z_2 = 3 - 4i.$$

Najpierw rozszerzamy ułamek przez sprzężenie mianownika:

$$\frac{z_1}{z_2} = \frac{1+2i}{3-4i} = \frac{1+2i}{3-4i} \cdot \frac{3+4i}{3+4i}.$$

Następnie wykonujemy mnożenie w liczniku, a w mianowniku stosujemy wzór skróconego mnożenia:

$$\ldots = \frac{(1+2i)(3+4i)}{(3-4i)(3+4i)} = \frac{1 \cdot 3 + 1 \cdot 4i + 2i \cdot 3 + 2i \cdot 4i}{3^2 - (4i)^2}.$$

Wykonujemy rachunki i zapisujemy wynik w postaci algebraicznej (3.1):

$$\ldots = \frac{3+4i+6i-8}{9+16} = \frac{-5+10i}{25} = -\frac{1}{5}+\frac{2}{5}i.$$

W **Sage** stosujemy operator dzielenia /:

```
z1 = 1 + 2*i
z2 = 3 - 4*i
print('Iloraz liczb zespolonych:', z1/z2)
Out: Iloraz liczb zespolonych: 2/5*I - 1/5
```

3.1.3. Postać wykładnicza i postać trygonometryczna

Definicja. Postacią wykładniczą liczby zespolonej $z \in \mathbb{C}$ nazywamy zapis:

$$z = |z| \cdot e^{i\varphi}, \tag{3.4}$$

gdzie $|z|$ jest modułem, a φ – argumentem liczby z.

Przykład 3.10
Zapiszemy w postaci wykładniczej liczbę zespoloną:

$$z = 1 + i.$$

Aby przedstawić liczbę zespoloną w postaci wykładniczej, potrzebujemy dwóch wielkości – modułu i argumentu. Moduł wyznaczamy na podstawie wzoru (3.3):

$$|z| = \sqrt{1^2 + 1^2} = \sqrt{2}.$$

Argument możemy natomiast odczytać z rysunku. Promień wodzący liczby zespolonej z pokrywa się z przekątną kwadratu o wierzchołkach w punktach $(0,0)$, $(1,0)$, $(1,1)$ i $(0,1)$. Jest on zatem nachylony do osi rzeczywistej pod kątem 45°, czyli:

$$\varphi = \frac{\pi}{4}.$$

W **Sage** czynności te wykonujemy za pomocą funkcji **abs()** oraz **arg()**. Wypisanie dwóch zmiennych po przecinku w ostatniej linii powoduje wyświetlenie na ekranie obu wartości jednocześnie:

```
z = 1 + i
modul = abs(z)
phi = arg(z)
modul, phi
Out: (sqrt(2), 1/4*pi)
```

Teraz możemy zapisać liczbę z w postaci wykładniczej (3.4):

$$z = \sqrt{2} \cdot e^{i\frac{\pi}{4}}.$$

Stosujemy argument hold=True wewnątrz funkcji **exp()**, aby wartość $e^{i\frac{\pi}{4}}$ nie była wyliczana, a jedynie została wypisana na ekran:

```
postac_wykladnicza = modul * exp(i*phi, hold=True)
print('Postać wykładnicza:')
show( LatexExpr('z='), postac_wykladnicza )
Out: z = √2e^(1/4 iπ)
```

Definicja. Postacią trygonometryczną liczby zespolonej $z \in \mathbb{C}$ nazywamy zapis:

$$z = |z| \cdot (\cos\varphi + i\sin\varphi), \tag{3.5}$$

gdzie $|z|$ jest modułem, a φ – argumentem liczby z.

Przykład 3.11

Zapiszemy w postaci trygonometrycznej liczbę zespoloną:

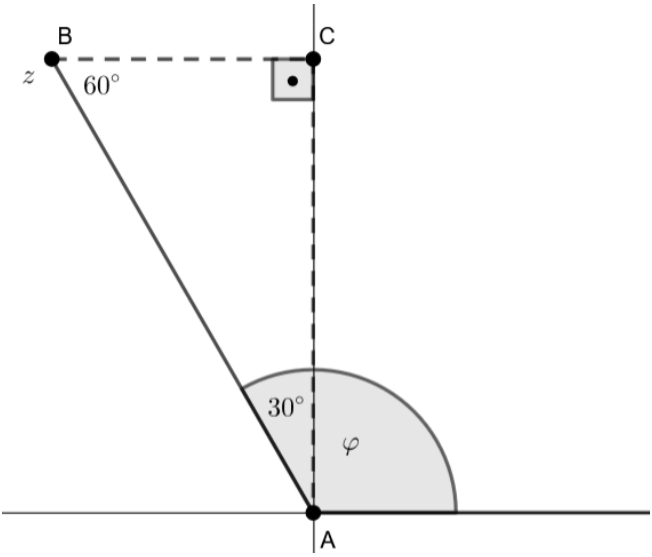
$$z = -\frac{1}{2} + i\frac{\sqrt{3}}{2}.$$

Aby zapisać liczbę zespoloną w postaci trygonometrycznej, potrzebujemy dokładnie tych samych wielkości co w przypadku postaci wykładniczej, czyli modułu i argumentu. Moduł ponownie wyznaczamy ze wzoru (3.3) jako:

$$|z| = \sqrt{\left(-\frac{1}{2}\right)^2 + \left(\frac{\sqrt{3}}{2}\right)^2} = 1.$$

Argument możemy natomiast odczytać z rysunku 3.6. Zauważmy, że miary kątów trójkąta o wierzchołkach w punktach $A = (0,0), B = \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$ i $C = \left(0, \frac{\sqrt{3}}{2}\right)$ wynoszą odpowiednio 30°, 60° i 90°. Kąt φ znajduje się w drugiej ćwiartce, więc argument główny jest o 30° większy niż kąt prosty 90°, stąd:

$$\varphi = \frac{\pi}{2} + \frac{\pi}{6} = \frac{2\pi}{3}.$$



Rysunek 3.6. Ilustracja liczby $z = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$ na płaszczyźnie zespolonej

W tym przykładzie funkcje **abs()** i **arg()** nie zapisują wyniku w najprostszej postaci, dlatego należy zastosować funkcję **simplify()**:

```
z = -1/2 + i*sqrt(3)/2
modul = abs(z).simplify()
```

```
phi = arg(z).simplify()
modul, phi
Out: (1, 2/3*pi)
```

Teraz możemy już przedstawić liczbę z w postaci trygonometrycznej (3.5):

$$z = \cos\frac{2\pi}{3} + i\sin\frac{2\pi}{3}.$$

Tutaj ponownie stosujemy argument hold=True:

```
postac_trygonometryczna = modul * (cos(phi, hold=True) +
 i * sin(phi, hold=True))
print('Postać trygonometryczna:')
show( LatexExpr('z='), postac_trygonometryczna )
Out: z = cos(2/3 π) + i sin(2/3 π)
```

Warto zauważyć, że obie postaci – wykładnicza (3.4) i trygonometryczna (3.5) – reprezentują tę samą liczbę zespoloną z. Możemy je zatem ze sobą porównać:

$$|z| \cdot e^{i\varphi} = |z| \cdot (\cos\varphi + i\sin\varphi).$$

W ten sposób otrzymujemy tzw. wzór Eulera:

$$e^{i\varphi} = \cos\varphi + i\sin\varphi. \tag{3.6}$$

Spróbujmy teraz pomnożyć dwie liczby zespolone zapisane w postaci wykładniczej, czyli:

$$z_1 = |z_1| \cdot e^{i\varphi_1} \quad \text{oraz} \quad z_2 = |z_2| \cdot e^{i\varphi_2}.$$

Otrzymujemy zależność:

$$z_1 \cdot z_2 = |z_1| \cdot e^{i\varphi_1} \cdot |z_2| \cdot e^{i\varphi_2} = |z_1||z_2| \cdot e^{i(\varphi_1+\varphi_2)}. \tag{3.7}$$

Wniosek. Przy mnożeniu liczb zespolonych mnożymy moduły i dodajemy argumenty.

Przykład 3.12
Przedstawimy interpretację graficzną mnożenia liczb zespolonych:

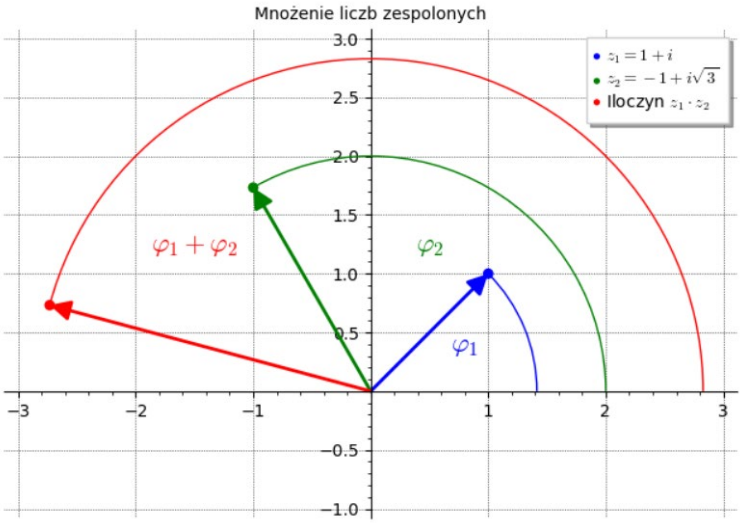
$$z_1 = 1 + i \quad \text{oraz} \quad z_2 = -1 + i\sqrt{3}.$$

Zaznaczamy na rysunku promienie wodzące, liczby zespolone i łuki oraz umieszczamy przy liczbach zespolonych tekst, podobnie jak w poprzednich przykładach. Efekt działania poniższego kodu przedstawiony został na rysunku 3.7.

```
z1 = 1 + i
z2 = -1 + i*sqrt(3)

rys = plot( vector(z1), gridlines=True,
 title='Mnożenie liczb zespolonych')
rys += plot( vector(z2.n()), color='green' )
rys += plot( vector(z1*z2.n()), color='red' )
rys += point( z1, size=40, zorder=5,
 legend_label='$z_1=1+i$' )
rys += point( z2, color='green', size=40, zorder=5,
 legend_label=r'$z_2=-1+i\sqrt{3}$' )
rys += point( z1*z2, color='red', size=40, zorder=5,
 legend_label=r'Iloczyn $z_1\cdot z_2$' )
rys += arc( (0, 0), abs(z1),
 sector=(0, arg(z1)) )
rys += arc( (0, 0), abs(z2),
 sector=(0, arg(z2)), color='green' )
rys += arc( (0, 0), abs(z1*z2),
 sector=(0, arg(z1*z2)), color='red' )
rys += text( r'$\varphi_1$', (0.8, 0.4),
 fontsize=16 )
rys += text( r'$\varphi_2$', (0.5, 1.25),
 color='green', fontsize=16 )
rys += text( r'$\varphi_1+\varphi_2$', (-1.5, 1.25),
 color='red', fontsize=16 )
```

```
rys.axes_range(-3, 3, -1, 3)
rys
```

Wniosek. Mnożenie liczby zespolonej $z \in \mathbb{C}$ przez liczbę zespoloną $e^{i\theta}$ powoduje jej obrót o kąt θ przeciwnie do ruchu wskazówek zegara wokół początku układu współrzędnych.



Rysunek 3.7. Interpretacja graficzna mnożenia dwóch liczb zespolonych

Przykład 3.13

Wykonamy obrót liczby zespolonej

$$z = 1 + i\sqrt{3}$$

o kąt $\theta = \frac{\pi}{3}$ przeciwnie do ruchu wskazówek zegara.

Zgodnie z powyższym wnioskiem wykonujemy działanie:

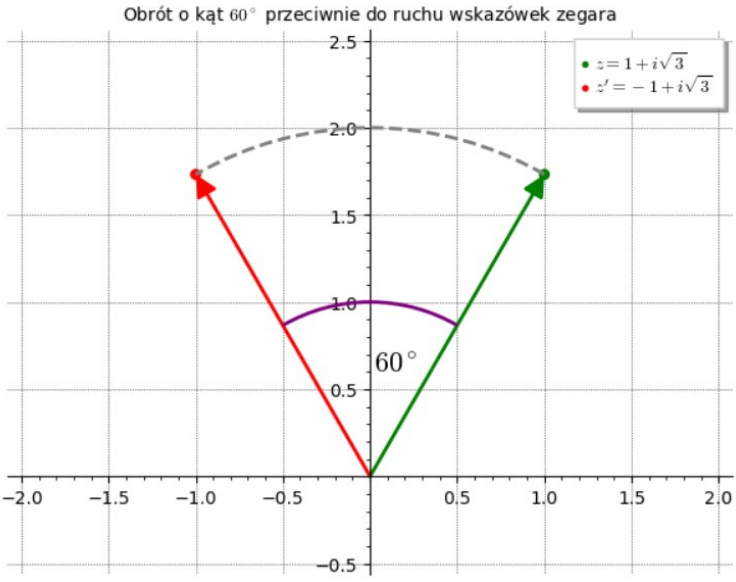
$$z \cdot e^{i\frac{\pi}{3}}.$$

W **Sage** stosujemy kod:

```
z = 1 + i*sqrt(3)
theta = pi/3
obrot = z * exp(i*theta)
print( 'Liczba zespolona po obrocie:', expand(obrot) )
Out: Liczba zespolona po obrocie: I*sqrt(3) – 1
```

Interpretację graficzną obrotu przedstawiamy na rysunku 3.8, wykorzystując instrukcje:

```
rys = plot( vector(z.n()), color='green', gridlines=True,
 title=r'Obrót o kąt $60^\circ$ przeciwnie ' +
 'do ruchu wskazówek zegara' )
rys += plot( vector(obrot.n()), color='red' )
rys += point( z, color='green', size=40, zorder=5,
 legend_label=r'$z=1+i\sqrt{3}$' )
rys += point( obrot, color='red', size=40, zorder=5,
 legend_label=r"$z'=-1+i\sqrt{3}$" )
rys += arc( (0, 0), 1, sector=(arg(z), arg(obrot)),
 thickness=2, color='purple' )
rys += arc( (0, 0), abs(z), sector=(arg(z), arg(obrot)),
 thickness=2, color='gray', linestyle='--' )
rys += text( r'$60^\circ$', (0.15, 0.65),
 color='black', fontsize=16 )
rys.axes_range(-2, 2, -0.5, 2.5)
rys
```



Rysunek 3.8. Wizualizacja obrotu liczby zespolonej $1 + i\sqrt{3}$ o kąt 60° przeciwnie do ruchu wskazówek zegara

Początkowa liczba zespolona $z = 1 + i\sqrt{3}$ została zaznaczona kolorem zielonym, natomiast liczbę zespoloną z', uzyskaną po wykonaniu obrotu, narysowano kolorem czerwonym. Funkcja **arc()** dodaje łuki okręgu, natomiast funkcja **text()** – napis 60° we wskazanym położeniu. Warto zauważyć, że w drugim wywołaniu funkcji **point()**, przy określaniu legendy, należy zastosować cudzysłów (""), ponieważ tekst zawiera już apostrof (').

Teraz możemy lepiej zrozumieć zależność z przykładu 3.8. Ponieważ moduł jednostki urojonej jest równy 1, iloczyn $z \cdot i$ nie zmienia wartości modułu liczby z. Jednakże argument główny liczby i wynosi $\frac{\pi}{2}$, co oznacza, że mnożenie przez jednostkę urojoną i powoduje obrót liczby z o kąt 90° przeciwnie do ruchu wskazówek zegara. Rzeczywiście, na podstawie wzoru Eulera (3.6) otrzymujemy potwierdzenie:

$$e^{i\frac{\pi}{2}} = \cos\frac{\pi}{2} + i\sin\frac{\pi}{2} = 0 + i \cdot 1 = i.$$

Podobnie możemy wykonywać potęgowanie liczby zespolonej $z \in \mathbb{C}$ zapisanej w postaci wykładniczej (3.4):

$$z^n = \left(|z| \cdot e^{i\varphi}\right)^n = |z|^n \cdot e^{in\varphi}. \tag{3.8}$$

Po przekształceniu wyniku do postaci trygonometrycznej (3.5) uzyskujemy tzw. wzór de Moivre'a:

$$z^n = |z|^n \cdot (\cos(n\varphi) + i\sin(n\varphi)).$$

Spróbujmy teraz wykonać dzielenie dwóch liczb zespolonych zapisanych w postaci wykładniczej, czyli:

$$z_1 = |z_1| \cdot e^{i\varphi_1} \quad \text{oraz} \quad z_2 = |z_2| \cdot e^{i\varphi_2}.$$

Z poniższego rachunku wynika, że przy dzieleniu dwóch liczb zespolonych dzielimy ich moduły i odejmujemy argumenty:

$$\frac{z_1}{z_2} = \frac{|z_1| \cdot e^{i\varphi_1}}{|z_2| \cdot e^{i\varphi_2}} = \frac{|z_1|}{|z_2|} \cdot e^{i(\varphi_1 - \varphi_2)}.$$

Wniosek. Dzielenie liczby zespolonej $z \in \mathbb{C}$ przez liczbę zespoloną $e^{i\theta}$ (lub równoważnie – mnożenie przez liczbę $e^{-i\theta}$) powoduje obrót liczby z o kąt θ zgodnie z ruchem wskazówek zegara wokół początku układu współrzędnych.

Przykład 3.14
Wykonamy obrót liczby zespolonej

$$z = 1 + i$$

o kąt $\theta = \frac{\pi}{6}$ zgodnie z ruchem wskazówek zegara.

Na podstawie wcześniejszego wniosku wykonujemy działanie:

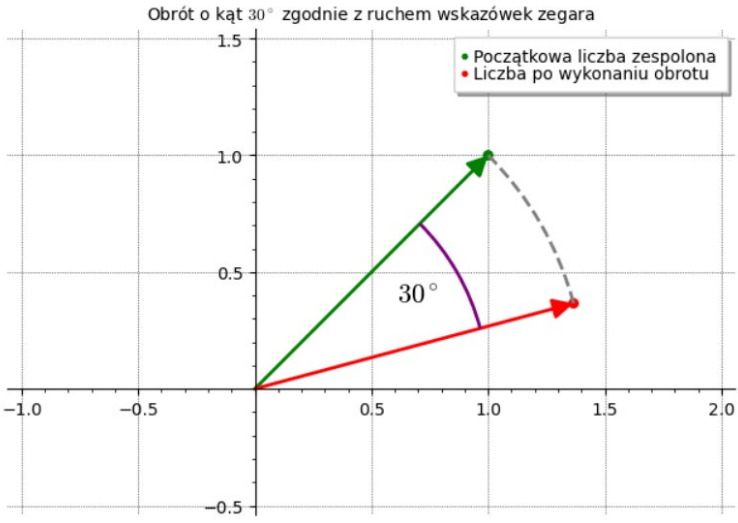
$$z \cdot e^{-i\frac{\pi}{6}}.$$

W **Sage** piszemy:

```
z = 1 + i
theta = pi/6


obrot = z * exp(-i*theta)
print( 'Liczba zespolona po obrocie:', expand(obrot) )
Out: Liczba zespolona po obrocie:
 (1/2*I + 1/2)*sqrt(3) - 1/2*I + 1/2
```

Przedstawiamy wizualizację tego obrotu na płaszczyźnie zespolonej:

```
rys = plot( vector(z.n()), color='green', gridlines=True,
 title=r'Obrót o kąt $30^\circ$ zgodnie ' +
 'z ruchem wskazówek zegara' )
rys += plot( vector(obrot.n()), color='red' )
rys += point( z, color='green', size=40, zorder=5,
 legend_label='Początkowa liczba zespolona' )
rys += point( obrot, color='red', size=40, zorder=5,
 legend_label='Liczba po wykonaniu obrotu' )
rys += arc( (0, 0), 1, sector=(arg(z), arg(obrot)),
 thickness=2, color='purple' )
rys += arc( (0, 0), abs(z), sector=(arg(z), arg(obrot)),
 thickness=2, color='gray', linestyle='--' )
rys += text( r'$30^\circ$', (0.7, 0.4),
 color='black', fontsize=16 )
rys.axes_range(-1, 2, -0.5, 1.5)
rys
```



Rysunek 3.9. Obrót liczby zespolonej $1 + i$ o kąt 30° zgodnie z ruchem wskazówek zegara

Wynik przedstawiono na rysunku 3.9. Podsumujemy jeszcze raz dotychczasowe wnioski. Mnożenie liczby zespolonej $z \in \mathbb{C}$ przez liczbę:

- $e^{i\theta}$ powoduje obrót liczby z o kąt θ przeciwnie do ruchu wskazówek zegara;
- $e^{-i\theta}$ powoduje obrót liczby z o kąt θ zgodnie z ruchem wskazówek zegara.

3.2. Zaznaczanie obszarów na płaszczyźnie zespolonej

Teraz dowiemy się, jak można zaznaczać obszary na płaszczyźnie zespolonej. Proces ten polega na wyróżnianiu miejsc na płaszczyźnie, które odpowiadają liczbom zespolonym spełniającym określone warunki.

W tym rozdziale przedstawimy kilka technik rozwiązywania tego typu zagadnień, m.in.:

- interpretację geometryczną nierówności z modułem i liczbą;
- interpretację geometryczną nierówności z dwoma modułami;
- algebraiczne wyznaczanie nierówności opisujących obszar;
- interpretację geometryczną nierówności z argumentem.

Warto podkreślić, że rozwiązywanie tego rodzaju zadań rozwija umiejętności kreatywnego i niestandardowego myślenia. Dzięki temu przygotowujemy się do radzenia sobie z nietypowymi problemami, które mogą się pojawić w przyszłości. Zatem, nawet jeśli praktyczne zastosowania tych zagadnień nie są teraz widoczne, warto potraktować je jako ciekawe wyzwanie intelektualne – w odpowiednim momencie może to przynieść nieoczekiwane korzyści.

3.2.1. Interpretacja geometryczna modułu

Rozpoczniemy od analizy warunków z modułem na płaszczyźnie zespolonej. Wyrażenie

$$|z|$$

reprezentuje odległość liczby zespolonej $z \in \mathbb{C}$ od początku układu współrzędnych.

Przykład 3.15

Zaznaczymy na płaszczyźnie zespolonej zbiory:

a) $A = \{z \in \mathbb{C}: |z| = 2\}$;

b) $B = \{z \in \mathbb{C}: |z| \leq 1\}$;

c) $C = \{z \in \mathbb{C}: |z| > 3\}$.

a) Warunek postaci

$$|z| = r$$

spełniają wszystkie liczby zespolone, których odległość od początku układu współrzędnych jest równa r. Zatem zbiór

$$\{z \in \mathbb{C}: |z| = r\}$$

jest okręgiem o środku w punkcie $(0, 0)$ i promieniu r. Ten sam wniosek można wyciągnąć na podstawie obliczeń. Po wstawieniu w miejsce zmiennej z jej postaci algebraicznej, czyli $z = x + yi$, gdzie $x, y \in \mathbb{R}$, oraz zastosowaniu wzoru na moduł (3.3) uzyskujemy:

$$|x + yi| = r \rightarrow \sqrt{x^2 + y^2} = r \rightarrow x^2 + y^2 = r^2.$$

Otrzymujemy równanie okręgu $O\big((0, 0), r\big)$.

Wobec tego zbiór

$$A = \{z \in \mathbb{C}: |z| = 2\}$$

reprezentuje okrąg o środku w punkcie $(0, 0)$ i promieniu równym 2. Aby go narysować, stosujemy następujący kod:

```
var('x y')
z = x + y*i
A = region_plot( abs(z) == 2, (x, -3, 3), (y, -3, 3),
 bordercol='black',
 title='Zbiór $A$', gridlines=True )
```

Definiujemy tutaj zmienne x i y jako symbole oraz stosujemy postać algebraiczną dla zmiennej z. Wystarczy zastosować ten zapis tylko raz – system **SageMath** zachowuje w pamięci wcześniej zdefiniowane zmienne, dzięki czemu można z nich korzystać przy tworzeniu kolejnych rysunków. Zauważmy, że funkcja **region_plot()** pozwala także zaznaczać na płaszczyźnie warunki w postaci równości.

b) Warunek postaci

$$|z| < r$$

spełniają wszystkie liczby zespolone, których odległość od początku układu współrzędnych jest mniejsza niż r.

Zatem zbiór

$$\{z \in \mathbb{C}: |z| < r\}$$

opisuje koło otwarte (bez brzegu) $K\big((0,0),r\big)$ o środku w punkcie $(0,0)$ i promieniu r. W sytuacji, gdy w warunku występuje słaba nierówność (czyli $|z| \leq r$), obszar stanowi koło domknięte $\overline{K}\big((0,0),r\big)$.

Stąd wynika, że zbiór

$$B = \{z \in \mathbb{C}: |z| \leq 1\}$$

przedstawia koło jednostkowe. Jego wizualizację tworzymy w następujący sposób:

```
B = region_plot( abs(z) < 1, (x, -2, 2), (y, -2, 2),
 incol='cyan', bordercol='black',
 title='Zbiór $B$', gridlines=True )
```

c) Warunek postaci

$$|z| > r$$

spełniają wszystkie liczby zespolone, których odległość od początku układu współrzędnych jest większa niż r. Tym samym zbiór

$$\{z \in \mathbb{C}: |z| > r\}$$

przedstawia płaszczyznę bez koła o środku w punkcie $(0,0)$ i promieniu r. Zastosowanie słabej nierówności w warunku ($|z| \geq r$) sprawia, że obszar ten bierzemy razem z brzegiem.

Wobec tego zbiór

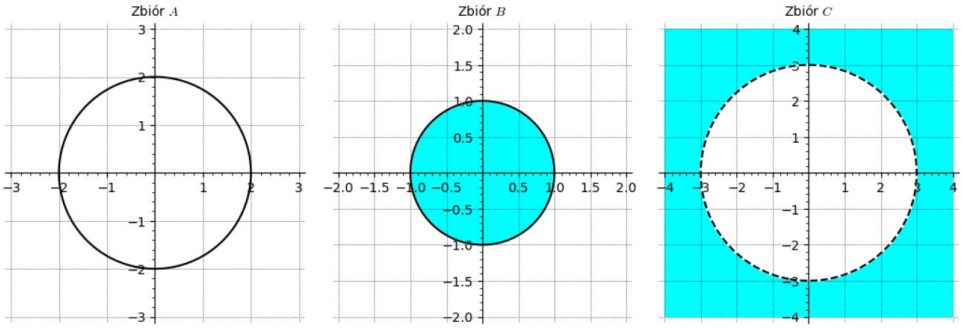
$$C = \{z \in \mathbb{C}: |z| > 3\}$$

jest płaszczyzną bez koła o promieniu 3. Do jego narysowania ponownie używamy funkcji **region_plot()**. Przekazujemy silną nierówność `abs(z) > 3` oraz ustawiamy przerywaną linię brzegu za pomocą argumentu `borderstyle='--'`:

```
C = region_plot( abs(z) > 3, (x, -4, 4), (y, -4, 4),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $C$', gridlines=True )
```

Na rysunku 3.10 przedstawiamy wszystkie trzy ilustracje, korzystając z poleceń:

```
A.show(figsize=5)
B.show(figsize=5)
C.show(figsize=5)
```



Rysunek 3.10. Obszary na płaszczyźnie zespolonej spełniające warunki: $|z| = 2$, $|z| \leq 1$ i $|z| > 3$

Wyrażenie postaci

$$|z - z_0|$$

określa odległość liczby zespolonej $z \in \mathbb{C}$ od ustalonej liczby zespolonej $z_0 \in \mathbb{C}$.

Przykład 3.16

Narysujemy zbiory:

a) $D = \{z \in \mathbb{C}: |z + 1 - 2i| = 3\}$;

b) $E = \{z \in \mathbb{C}: |z + i| < 2\}$;

c) $F = \{z \in \mathbb{C}: |z - 3 + i| \geq 4\}$.

a) Przekształcamy moduł występujący w zbiorze D do postaci:

$$|z + 1 - 2i| = |z - (-1 + 2i)|.$$

Warunek ten opisuje odległość liczby $z \in \mathbb{C}$ od liczby zespolonej $z_0 = -1 + 2i$. Wobec tego zbiór

$$D = \{z \in \mathbb{C}: |z - (-1 + 2i)| = 3\}$$

jest okręgiem $O\big((-1, 2), 3\big)$ o środku w punkcie $(-1, 2)$ i promieniu 3. Zaznaczamy go na rysunku za pomocą następującej instrukcji:

```
D = region_plot( abs(z+1-2*i) == 3, (x, -5, 3), (y, -2, 6),
 bordercol='black',
 title='Zbiór $D$', gridlines=True )
```

b) Zapisujemy zadany zbiór w postaci:

$$E = \{z \in \mathbb{C}: |z - (-i)| < 2\}.$$

Oznacza to, że obszar ten składa się z punktów, których odległość od liczby $-i$ jest mniejsza niż 2. Jest to więc koło otwarte $K\big((0,-1),2\big)$ o środku w punkcie $(0,-1)$ i promieniu 2. Przy okazji warto zauważyć, że:

- środek koła to liczba $z \in \mathbb{C}$, dla której wyrażenie w module przyjmuje wartość 0;
- promień koła jest równy liczbie występującej po symbolu porównania.

Zaznaczamy obszar E na płaszczyźnie, używając polecenia:

```
E = region_plot( abs(z+i) < 2, (x, -3, 3), (y, -4, 2),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $E$', gridlines=True )
```

c) Zastosujemy obserwację zawartą w poprzednim podpunkcie, aby przeprowadzić interpretację geometryczną kolejnego zbioru:

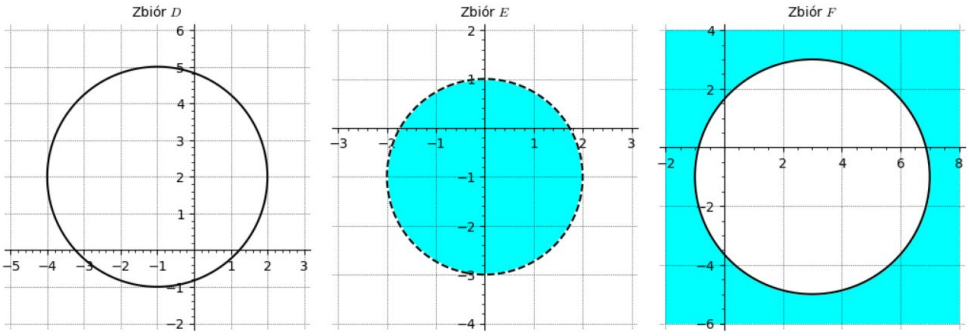
$$F = \{z \in \mathbb{C}: |z - 3 + i| \geq 4\}.$$

Wyrażenie w module zerowane jest dla liczby zespolonej $z_0 = 3 - i$. Zbiór F opisuje zatem wszystkie punkty, których odległość od liczby z_0 jest większa lub równa 4. Stąd wynika, że przedstawia on płaszczyznę bez koła otwartego o środku w punkcie $(3,-1)$ i promieniu równym 4. Graficzną reprezentację tworzymy ponownie za pomocą funkcji **region_plot()**:

```
F = region_plot( abs(z-3+i) > 4, (x, -2, 8), (y, -6, 4),
 incol='cyan', bordercol='black',
 title='Zbiór $F$', gridlines=True )
```

Wszystkie trzy zbiory prezentujemy na rysunku 3.11, stosując polecenia:

```
D.show(figsize=5)
E.show(figsize=5)
F.show(figsize=5)
```



Rysunek 3.11. Interpretacja geometryczna warunków z modułem i liczbą na płaszczyźnie zespolonej

Wyrażenie postaci

$$|z - z_1| = |z - z_2|$$

spełniają liczby zespolone $z \in \mathbb{C}$, których odległość od danych liczb zespolonych $z_1, z_2 \in \mathbb{C}$ jest taka sama. Oznacza to, że leżą one na symetralnej odcinka łączącego punkty z_1 i z_2, czyli prostej będącej zbiorem punktów równoodległych od jego końców.

Przykład 3.17

Na płaszczyźnie zespolonej zaznaczymy zbiory:

a) $G = \{z \in \mathbb{C}: |z - 1 - 2i| = |z + 3 + i|\}$;

b) $H = \{z \in \mathbb{C}: |z + 1 - i| < |z - 2 + 3i|\}$.

a) Przekształcamy wyrażenia w modułach występujące w definicji zbioru G:

$$|z - 1 - 2i| = |z - (1 + 2i)| \quad \text{oraz} \quad |z + 3 + i| = |z - (-3 - i)|.$$

Zgodnie z powyższym opisem zbiór

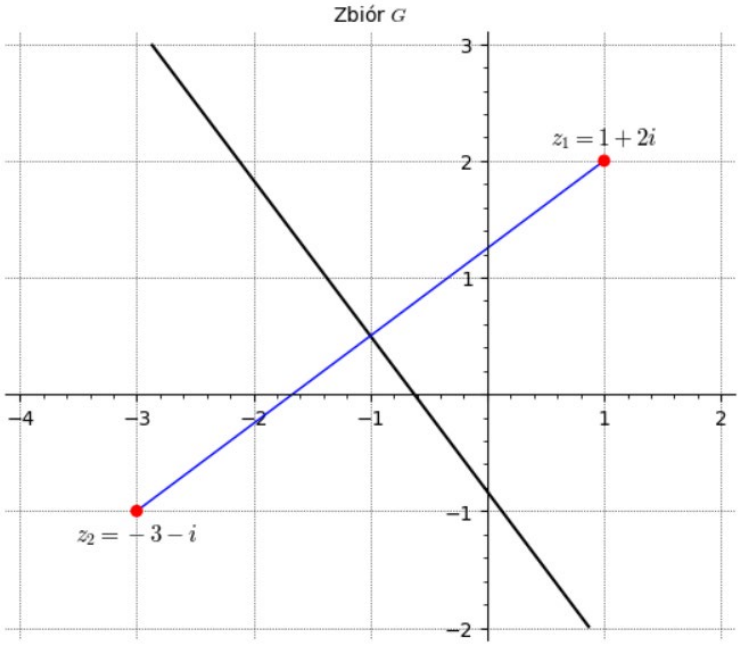
$$G = \{z \in \mathbb{C}: |z - (1 + 2i)| = |z - (-3 - i)|\}$$

składa się ze wszystkich punktów równoodległych od liczb $z_1 = 1 + 2i$ oraz $z_2 = -3 - i$. Jest to symetralna odcinka o końcach w punktach $(1, 2)$ i $(-3, -1)$. Aby ją narysować, stosujemy następujący kod:

```
rys = region_plot( abs(z-1-2*i) == abs(z+3+i), (x, -4, 2), (y, -2, 3),
 bordercol='black', title='Zbiór $G$', gridlines=True )
```

```
rys += points( [(1, 2), (-3, -1)], color='red', size=40, zorder=5 )
rys += line( [(1, 2), (-3, -1)] )
rys += text( '$z_1=1+2i$', (1, 2.2), color='black', fontsize=12 )
rys += text( '$z_2=-3-i$', (-3, -1.2), color='black', fontsize=12 )
rys
```

Uzyskaną symetralną zaznaczono kolorem czarnym na rysunku 3.12. W celu lepszej wizualizacji dodany został niebieski odcinek łączący punkty z_1 i z_2. Wartości określające zakresy i współrzędne w kodzie dobieramy metodą prób i błędów.



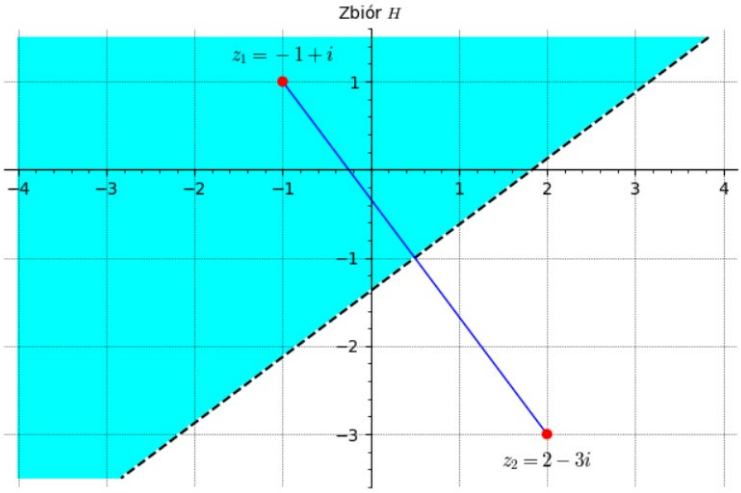
Rysunek 3.12. Zbiór G – symetralna niebieskiego odcinka o końcach w punktach $z_1 = 1 + 2i$ oraz $z_2 = -3 - i$

b) Zbiór

$$H = \{z \in \mathbb{C}: |z - (-1 + i)| < |z - (2 - 3i)|\}$$

składa się ze wszystkich punktów, których odległość od liczby $z_1 = -1 + i$ jest mniejsza niż odległość od liczby $z_2 = 2 - 3i$. Jest to zatem półpłaszczyzna leżąca po tej samej stronie symetralnej co punkt z_1. Zaznaczamy ten obszar na płaszczyźnie zespolonej za pomocą poniższego kodu:

```
rys = region_plot( abs(z+1-i) < abs(z-2+3*i), (x, -4, 4), (y, -3.5, 1.5),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $H$', gridlines=True )
rys += points( [(-1, 1), (2, -3)], color='red', size=40, zorder=5 )
rys += line( [(-1, 1), (2, -3)] )
rys += text( '$z_1=-1+i$', (-1, 1.3), color='black', fontsize=12 )
rys += text( '$z_2=2-3i$', (2, -3.3), color='black', fontsize=12 )
rys
```



Rysunek 3.13. Interpretacja geometryczna nierówności z dwoma modułami na płaszczyźnie zespolonej

Na rysunku 3.13 zbiór H przedstawiony został błękitnym kolorem. Analogicznie jak poprzednio dodajemy odcinek łączący punkty z_1 i z_2, aby zilustrować interpretację geometryczną. Warto zwrócić uwagę, że liczby z_1 i z_2 można też wyznaczyć za pomocą metody z przykładu 3.16 – wystarczy odczytać wartości zerujące poszczególne moduły.

3.2.2. Algebraiczne wyznaczanie nierówności opisującej obszar

Przejdziemy teraz do analizy nieco bardziej złożonych obszarów na płaszczyźnie zespolonej. W niektórych przypadkach interpretacja geometryczna okazuje się niewystarczająca – wówczas konieczne jest wyznaczenie odpowiednich ograniczeń w sposób analityczny.

Przykład 3.18

Na płaszczyźnie zespolonej narysujemy zbiory:

a) $A = \{z \in \mathbb{C}\colon \operatorname{Re}[(z+i)^2] \leq 0\}$;
b) $B = \{z \in \mathbb{C}\colon |z| < \operatorname{Im}(\overline{z} + zi)\}$;
c) $C = \{z \in \mathbb{C}\colon 1 \leq |z+2-i| < 3 \ \wedge\ -4 < \operatorname{Im}(iz) \leq -1\}$.

a) Gdy w definicji zbioru występują operacje $\operatorname{Re} z$, $\operatorname{Im} z$ lub sprzężenie $\overline{z}$ liczby $z \in \mathbb{C}$, zazwyczaj nie jesteśmy w stanie zastosować interpretacji geometrycznej. W takich przypadkach należy:
   - zapisać liczbę z w postaci algebraicznej, czyli $z = x + yi$ $(x, y \in \mathbb{R})$;
   - przekształcić warunek do odpowiedniej nierówności opisującej obszar.

Dla warunku ze zbioru A podnosimy do kwadratu[9] wyrażenie w nawiasie i otrzymujemy:

$$\operatorname{Re}[(x+yi+i)^2] \leq 0 \ \rightarrow\ \operatorname{Re}(x^2 - y^2 - 1 + 2xyi + 2xi - 2y) \leq 0.$$

Grupujemy uzyskany wynik, aby wyodrębnić część rzeczywistą i część urojoną wyrażenia w nawiasie:

$$\operatorname{Re}[(x^2 - y^2 - 2y - 1) + (2xy + 2x)i] \leq 0.$$

Następnie wybieramy część rzeczywistą tego wyrażenia – jest to pierwszy nawias bez jednostki urojonej i, ponieważ w postaci algebraicznej (3.1) zakładamy, że obie wartości x i y są liczbami rzeczywistymi.

Otrzymujemy zależność:

$$x^2 - y^2 - 2y - 1 \leq 0 \ \rightarrow\ x^2 - (y+1)^2 \leq 0 \ \rightarrow\ (x - y - 1)(x + y + 1) \leq 0.$$

[9] Stosujemy wzór skróconego mnożenia: $(a+b+c)^2 = a^2 + b^2 + c^2 + 2ab + 2ac + 2bc$.

Iloczyn dwóch liczb jest mniejszy lub równy 0, gdy liczby te mają przeciwne znaki lub są równe 0. Stąd uzyskujemy nierówności opisujące obszar:

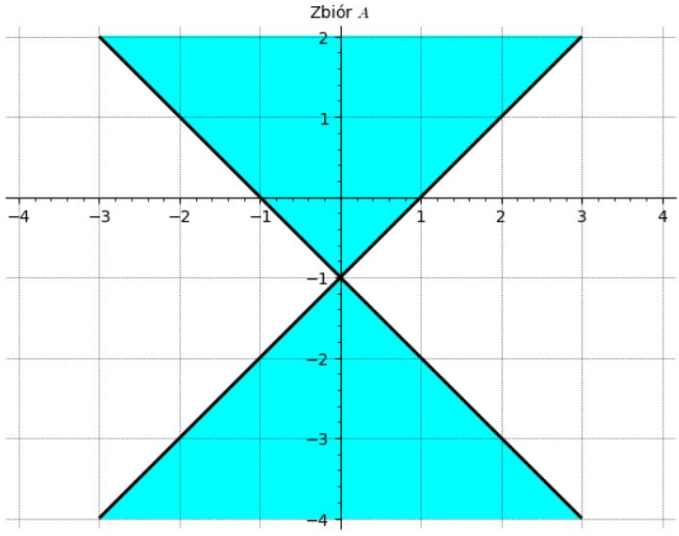
$$(x - y - 1 \geq 0 \ \wedge \ x + y + 1 \leq 0) \ \vee \ (x - y - 1 \leq 0 \ \wedge \ x + y + 1 \geq 0),$$

$$(y \leq x - 1 \ \wedge \ y \leq -x - 1) \ \vee \ (y \geq x - 1 \ \wedge \ y \geq -x - 1).$$

W **Sage** wystarczy przekazać zadane ograniczenie jako pierwszy argument funkcji **region_plot()**:

```
var('x y')
z = x + y*i
region_plot( real((z+i)^2) < 0, (x, -4, 4), (y, -4, 2),
 incol='cyan', bordercol='black',
 borderwidth=2, title='Zbiór $A$',
 gridlines=True, plot_points=200 )
```

Wynik przedstawiony został na rysunku 3.14. Dla lepszej widoczności zwiększamy grubość brzegu za pomocą argumentu borderwidth. Rysunki zawierające ostre krawędzie mogą być mniej dokładne, dlatego dodajemy większą wartość argumentu plot_points, co sprawia, że wyświetlany obszar jest bardziej precyzyjny.



Rysunek 3.14. Liczby $z \in \mathbb{C}$ na płaszczyźnie zespolonej spełniające warunek $\mathrm{Re}[(z+i)^2] \leq 0$

b) Dla zbioru

$$B = \{z \in \mathbb{C}: |z| < \mathrm{Im}(\overline{z} + zi)\}$$

ponownie stosujemy postać algebraiczną liczby zespolonej $z = x + yi$, gdzie $x, y \in \mathbb{R}$, oraz wykorzystujemy wzory na moduł (3.3) i sprzężenie (3.2):

$$|x + yi| < \mathrm{Im}\left[\, \overline{x + yi} + (x + yi)i \,\right] \rightarrow \sqrt{x^2 + y^2} < \mathrm{Im}(x - yi + xi - y).$$

Następnie grupujemy składniki po prawej stronie nierówności i wybieramy część urojoną:

$$\sqrt{x^2 + y^2} < \mathrm{Im}[(x - y) + (x - y)i] \rightarrow \sqrt{x^2 + y^2} < x - y.$$

Nierówność możemy równoważnie podnieść do kwadratu tylko wtedy, gdy obie strony są nieujemne (jeżeli zachodzi warunek $x - y < 0$, to nierówność jest sprzeczna). Podnosimy nierówność do kwadratu, zakładając dodatkowo, że prawa strona jest nieujemna:

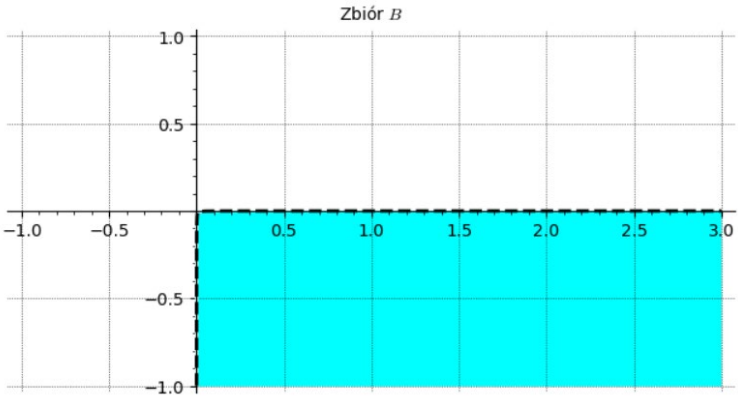
$$x - y \geq 0 \ \wedge \ x^2 + y^2 < x^2 - 2xy + y^2 \ \rightarrow \ y \leq x \ \wedge \ xy < 0.$$

Warunek $xy < 0$ oznacza, że liczby x i y mają przeciwne znaki. Rozwiązania znajdują się zatem w II lub IV ćwiartce. Jednak założenie $y \leq x$ powoduje, że bierzemy pod uwagę tylko rozwiązania z czwartej ćwiartki, czyli:

$$x > 0 \ \wedge \ y < 0.$$

Wynik uzyskany za pomocą poniższego kodu przedstawiony został na rysunku 3.15.

```
region_plot( abs(z) < imag(conjugate(z) + z*i), (x, -1, 3), (y, -1, 1),
 bordercol='black', incol='cyan', borderstyle='--', borderwidth=2,
 title='Zbiór $B$', gridlines=True )
```



Rysunek 3.15. Liczby zespolone $z \in \mathbb{C}$ spełniające nierówność $|z| < \mathrm{Im}(\,\overline{z} + zi\,)$

c) Zbiór

$$C = \{z \in \mathbb{C}\colon 1 \leq |z + 2 - i| < 3 \ \wedge \ -4 < \mathrm{Im}(iz) \leq -1\}$$

jest częścią wspólną dwóch warunków. Pierwszy z nich można zapisać jako:

$$1 \leq |z - (-2 + i)| < 3.$$

Nierówność ta opisuje wszystkie liczby $z \in \mathbb{C}$, których odległość od punktu $z_0 = -2 + i$ jest większa lub równa 1 i mniejsza niż 3. Geometrycznie tworzą one zatem pierścień o środku w punkcie $(-2, 1)$ z promieniem wewnętrznym 1 i zewnętrznym 3. Rysujemy go za pomocą poniższego kodu:

```
rys1 = region_plot( [1 < abs(z+2-i), abs(z+2-i) < 3], (x, -6, 2), (y, -3, 5),
 incol='cyan', bordercol='black',
 borderstyle='--', gridlines=True )
rys1 += region_plot( abs(z+2-i) == 1, (x, -6, 2), (y, -3, 5),
 bordercol='black', borderwidth=2 )
```

Pierścień ten jest wewnętrznie domknięty i zewnętrznie otwarty. Najpierw rysujemy oba brzegi linią przerywaną, a następnie dodajemy brzeg wewnętrzny nadpisany linią ciągłą. Część wspólną dwóch nierówności należy koniecznie umieścić po przecinku w liście będącej pierwszym argumentem funkcji **region_plot()**.

Drugie ograniczenie, czyli:

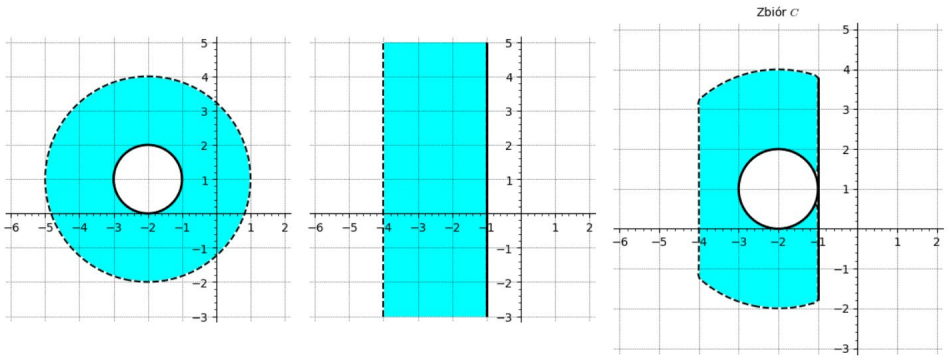
$$-4 < \mathrm{Im}(iz) \leq -1,$$

wymaga rozwiązania algebraicznego. Po podstawieniu postaci $z = x + yi$, gdzie $x, y \in \mathbb{R}$, otrzymujemy:

$$-4 < \mathrm{Im}[i(x + yi)] \leq -1 \;\rightarrow\; -4 < \mathrm{Im}(-y + xi) \leq -1 \;\rightarrow\; -4 < x \leq -1.$$

Jest to prawostronnie domknięty pionowy pas między prostymi pionowymi $x = -4$ oraz $x = -1$. Ponownie zaznaczamy brzeg linią przerywaną, a następnie nadpisujemy prawą stronę obszaru linią ciągłą, korzystając z niżej podanego kodu:

```
rys2 = region_plot( [-4 < imag(i*z), imag(i*z) < -1], (x, -6, 2), (y, -3, 5),
 incol='cyan', bordercol='black',
 borderstyle='--', gridlines=True )
rys2 += region_plot( imag(i*z) == -1, (x, -6, 2), (y, -3, 5),
 bordercol='black', borderwidth=2 )
```



Rysunek 3.16. Etapy konstrukcji zbioru C: pierścień, pionowy pas i ich przecięcie

Ostatecznie należy zaznaczyć liczby zespolone $z \in \mathbb{C}$ należące do obu obszarów jednocześnie. Na rysunku 3.16 przedstawiono końcowy wykres, który otrzymujemy poprzez odpowiednie połączenie fragmentów kodu:

```
rys3 = region_plot( [1 < abs(z+2-i), abs(z+2-i) < 3,
 -4 < imag(i*z), imag(i*z) < -1], (x, -6, 2), (y, -3, 5),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $C$', gridlines=True )
```

```
rys3 += region_plot( abs(z+2-i) == 1, (x, -6, 2), (y, -3, 5),
 bordercol='black', borderwidth=2 )
rys3 += region_plot( imag(i*z) == -1, (x, -6, 2), (y, -1.8, 3.8),
 bordercol='black', borderwidth=2 )
rys1.show(figsize=5)
rys2.show(figsize=5)
rys3.show(figsize=6)
```

3.2.3. Interpretacja geometryczna nierówności z argumentem

W systemie **SageMath** obowiązuje konwencja określająca główny argument liczby zespolonej jako wartość z przedziału $(-\pi, \pi]$. Powoduje to pojawienie się nieciągłości przy przejściu między II a III ćwiartką układu współrzędnych. W praktyce oznacza to, że w niektórych przypadkach konieczne jest podzielenie rysunku na dwie niezależne części, tak aby uniknąć problemów z wizualizacją. Poniższy przykład ilustruje możliwe podejścia do rozwiązania tego zagadnienia.

Przykład 3.19

Na płaszczyźnie zespolonej zaznaczymy zbiory:

a) $A = \left\{ z \in \mathbb{C}: \frac{\pi}{3} \leq \arg z < \frac{11\pi}{6} \right\}$;

b) $B = \left\{ z \in \mathbb{C}: 0 \leq \arg(-2z) \leq \frac{4\pi}{3} \right\}$;

c) $C = \left\{ z \in \mathbb{C}: \frac{\pi}{2} \leq \arg(z^4) < \pi \right\}$.

a) Zapisujemy podwójną nierówność jako przecięcie dwóch warunków:

$$\frac{\pi}{3} \leq \arg z < 2\pi \ \wedge \ 0 \leq \arg z < \frac{11\pi}{6}.$$

Argument jest określony z dokładnością do 2π, więc możemy odjąć tę wartość stronami w drugiej nierówności:

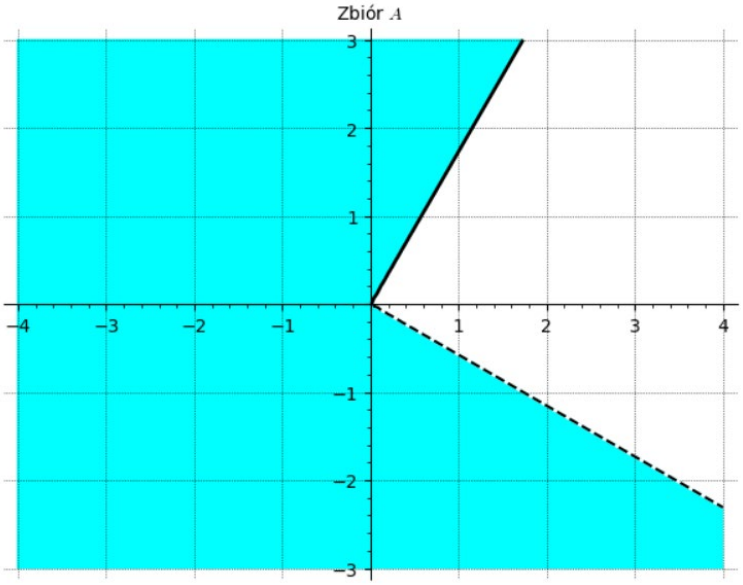
$$\frac{\pi}{3} \leq \arg z < 2\pi \ \wedge \ -2\pi \leq \arg z < -\frac{\pi}{6}.$$

Teraz przechodzimy na zapis w konwencji obowiązującej w **Sage**:

$$\frac{\pi}{3} \leq \operatorname{Arg} z \leq \pi \ \vee \ -\pi < \operatorname{Arg} z < -\frac{\pi}{6}.$$

Aby zaznaczyć ten obszar na rysunku, stosujemy następujące polecenia:

```
var('x y')
z = x + y*i

rys = region_plot( pi/3 < arg(z), (x, -4, 4), (y, 0, 4),
 incol='cyan', bordercol='black',
 borderwidth=2, title='Zbiór $A$' )
rys += region_plot( arg(z) < -pi/6, (x, -4, 4), (y, -4, -.01),
 incol='cyan', bordercol='black',
 borderstyle='--', gridlines=True )
rys
```



Rysunek 3.17. Liczby na płaszczyźnie zespolonej spełniające nierówność $\frac{\pi}{3} \leq \arg z < \frac{11\pi}{6}$

Wynik działania powyższego kodu przedstawiono na rysunku 3.17. Aby rozwiązać problem nieciągłości między II a III ćwiartką, obszar dzielimy na dwie części:

- najpierw zaznaczamy fragment należący do I i II ćwiartki;
- następnie dodajemy pozostałą część należącą do III i IV ćwiartki.

Wykonujemy to poprzez wprowadzenie odpowiednich ograniczeń dla zakresu wyświetlania w pionie, na przykład: (y, 0, 4). Takie podejście rozwiązuje problem nieciągłości. Na rysunku widzimy, że rzeczywiście zaznaczono wszystkie liczby zespolone z, których argument należy do zadanego przedziału.

b) W przypadku zbioru

$$B = \left\{ z \in \mathbb{C}: 0 \leq \arg(-2z) \leq \frac{4\pi}{3} \right\}$$

najpierw skorzystamy ze wzoru (3.7), zgodnie z którym argument iloczynu liczb zespolonych równy jest sumie argumentów (z dokładnością do 2π). Dodatkowo argument główny liczby rzeczywistej ujemnej jest równy π. W związku z tym otrzymujemy:

$$\arg(-2z) = \arg[(-2) \cdot z] = \arg(-2) + \arg z + 2k\pi = \pi + \arg z + 2k\pi,$$

gdzie $k \in \mathbb{Z}$.

Po podstawieniu tej wartości do początkowej nierówności uzyskujemy zależność:

$$0 \leq \pi + \arg z + 2k\pi \leq \tfrac{4\pi}{3},$$

którą przekształcamy[10] do postaci:

$$-\pi + 2k\pi \leq \arg z \leq \frac{\pi}{3} + 2k\pi.$$

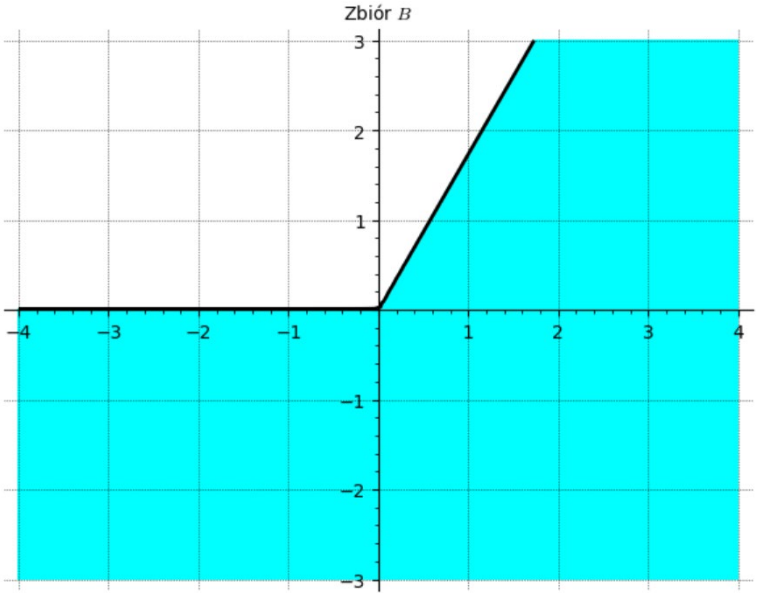
Następnie podstawiamy wszystkie możliwe wartości parametru $k \in \mathbb{Z}$, dla których argument mieści się w przedziale $(-\pi, \pi]$. W naszym przypadku warunek ten spełnia tylko wartość $k = 0$. Po podstawieniu jej do nierówności otrzymujemy ograniczenie:

$$-\pi \leq \operatorname{Arg} z \leq \frac{\pi}{3}.$$

Ponieważ w tym przypadku nie występuje przejście przez punkt nieciągłości, możemy zastosować pojedynczą komendę **region_plot()**. Na rysunku 3.18 przedstawiona została interpretacja zbioru B będąca efektem działania poniższego kodu:

[10] Znak wyrażenia $2k\pi$ nie ma znaczenia, ponieważ rozwiązania dla $k \in \mathbb{Z}$ ułożone są symetrycznie względem zera.

```
region_plot( arg(z) < pi/3, (x, -4, 4), (y, -3, 3),
 incol='cyan', bordercol='black', borderwidth=2,
 title='Zbiór $B$', gridlines=True )
```



Rysunek 3.18. Liczby na płaszczyźnie zespolonej spełniające warunek $0 \leq \arg(-2z) \leq \frac{4\pi}{3}$

c) Na koniec przeanalizujemy sposób przedstawienia zbioru:

$$C = \left\{ z \in \mathbb{C}\colon \frac{\pi}{2} \leq \arg(z^4) < \pi \right\}.$$

Ponownie korzystamy z własności argumentu iloczynu liczb zespolonych (3.7), mianowicie:

$$\arg(z^4) = \arg(z \cdot z \cdot z \cdot z) = \arg z + \arg z + \arg z + \arg z + 2k\pi,$$

gdzie $k \in \mathbb{Z}$.

W odniesieniu do nierówności otrzymujemy warunek w postaci:

$$\frac{\pi}{2} \leq 4 \arg z + 2k\pi < \pi,$$

który zapisujemy równoważnie jako:

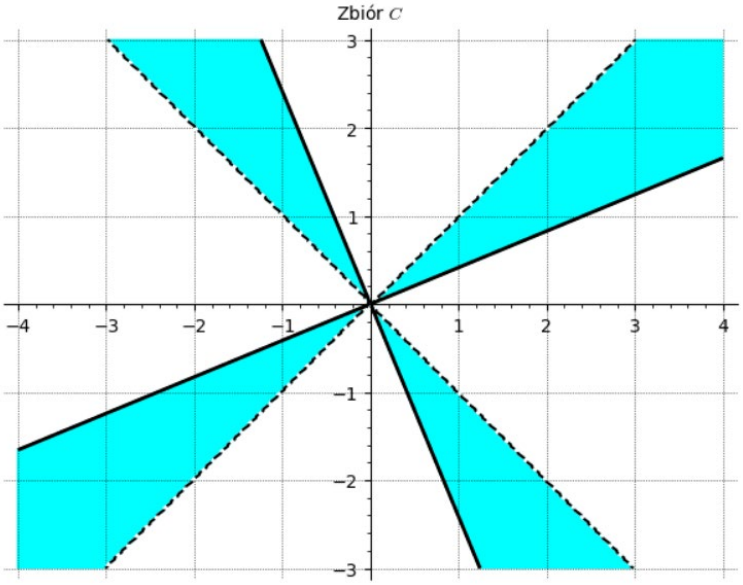
$$\frac{\pi}{8}+\frac{1}{2}k\pi \leq \arg z < \frac{\pi}{4}+\frac{1}{2}k\pi.$$

Argument będzie znajdował się w przedziale $(-\pi,\pi]$ dla parametrów $k \in \mathbb{Z}$ należących do zbioru $\{-2,-1,0,1\}$. W ten sposób otrzymujemy szukane ograniczenia:

$$-\frac{7\pi}{8} \leq \operatorname{Arg} z < -\frac{3\pi}{4} \vee -\frac{3\pi}{8} \leq \operatorname{Arg} z < -\frac{\pi}{4}$$
$$\vee \frac{\pi}{8} \leq \operatorname{Arg} z < \frac{\pi}{4} \vee \frac{5\pi}{8} \leq \operatorname{Arg} z < \frac{3\pi}{4}.$$

Wykonujemy wizualizację tego obszaru na płaszczyźnie zespolonej (rysunek 3.19):

```
rys = region_plot( [pi/2 < arg(z^4), arg(z^4) < pi], (x, -4, 4), (y, -3, 3),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $C$', plot_points=300 )
rys += region_plot( arg(z) == pi/8, (x, 0, 4), (y, 0, 3),
 bordercol='black', borderwidth=2 )
rys += region_plot( arg(z) == 5*pi/8, (x, -4, 0), (y, 0, 3),
 bordercol='black', borderwidth=2 )
rys += region_plot( arg(z) == -7*pi/8, (x, -4, -.01), (y, -3, -.01),
 gridlines=True, bordercol='black', borderwidth=2 )
rys += region_plot( arg(z) == -3*pi/8, (x, 0, 4), (y, -3, 0),
 bordercol='black', borderwidth=2 )
rys
```



Rysunek 3.19. Liczby na płaszczyźnie zespolonej spełniające warunek $\frac{\pi}{2} \leq \arg(z^4) < \pi$

3.3. Pierwiastek z liczby zespolonej

W zbiorze liczb rzeczywistych $\mathbb{R}$ pierwiastek z liczby jest pojedynczą wartością rzeczywistą. Dodatkowo pierwiastki parzystego stopnia możemy wyciągać tylko z liczb nieujemnych. Sytuacja zmienia się w ciele liczb zespolonych $\mathbb{C}$, gdzie pierwiastkowanie jest możliwe dla dowolnej liczby zespolonej. Co więcej, przyjmujemy, że pierwiastek n-tego stopnia z niezerowej liczby zespolonej z stanowi zbiór n-elementowy:

$$\sqrt[n]{z} = \{\omega_0, \omega_1, \omega_2, \dots, \omega_{n-1}\}, \quad \text{gdzie } \omega_k^n = z \text{ dla } k = 0, 1, 2, \dots, n-1.$$

Pierwiastki zespolone mają również interesującą interpretację geometryczną. Punkty ω_k tworzą wierzchołki n-kąta foremnego wpisanego w okrąg o środku w punkcie $(0, 0)$. W szczególnym przypadku, $n = 2$, otrzymujemy dwie liczby leżące naprzeciwko siebie na okręgu, symetryczne względem początku układu. Własność ta pozwala wykorzystywać geometrię płaszczyzny zespolonej do konstruowania w układzie współrzędnych wielokątów foremnych i innych figur symetrycznych.

3.3.1. Obliczanie pierwiastka z liczby zespolonej

Zaczniemy od przedstawienia metody wyznaczania pierwiastków kwadratowych w ciele liczb zespolonych. Pierwiastek kwadratowy z dodatniej liczby rzeczywistej obliczamy podobnie jak w przypadku zwykłych liczb rzeczywistych, z tą różnicą, że w zbiorze liczb zespolonych otrzymujemy dwa rozwiązania – tworzą one zbiór dwuelementowy, na przykład:

$$\sqrt{9} = \{-3, 3\}.$$

Wiemy, że dla jednostki urojonej i zachodzi równość $i^2 = -1$. Wynika stąd, że pierwiastek kwadratowy z liczby –1 jest równy:

$$\sqrt{-1} = \{-i, i\}.$$

Ta obserwacja pozwala nam w prosty sposób wyznaczać inne pierwiastki kwadratowe z liczb rzeczywistych ujemnych, na przykład:

$$\sqrt{-25} = \sqrt{25} \cdot \sqrt{-1} = 5 \cdot \{-i, i\} = \{-5i, 5i\}.$$

Warto zauważyć, że w każdym z powyższych przykładów oba pierwiastki są liczbami przeciwnymi. Okazuje się, że taką własność ma każdy pierwiastek kwadratowy. Dla dowolnej liczby $z \in \mathbb{C}$ jest to zbiór dwuelementowy:

$$\sqrt{z} = \{-\omega_0, \omega_0\},$$

gdzie liczbę $\omega_0 = x + yi$ (dla liczby z niebędącej liczbą rzeczywistą ujemną) wyznaczamy ze wzorów:

$$x = \sqrt{\frac{\operatorname{Re} z + |z|}{2}} \quad \text{oraz} \quad y = \frac{\operatorname{Im} z}{2x}. \tag{3.9}$$

Przykład 3.20

Obliczymy pierwiastek kwadratowy:

$$\sqrt{-16 + 30i}.$$

Dla liczby zespolonej $z = -16 + 30i$ odczytujemy część rzeczywistą $\mathrm{Re}\, z = -16$ i część urojoną $\mathrm{Im}\, z = 30$ oraz znajdujemy moduł na podstawie wzoru (3.3):

$$|z| = \sqrt{(-16)^2 + 30^2} = \sqrt{1156} = 34.$$

Następnie obliczamy jeden z pierwiastków $\omega_0 = x + yi$, korzystając ze wzorów (3.9):

$$x = \sqrt{\frac{-16+34}{2}} = \sqrt{9} = 3, \quad y = \frac{30}{2 \cdot 3} = 5.$$

W rezultacie otrzymujemy zbiór dwuelementowy:

$$\sqrt{-16 + 30i} = \{-3 - 5i, 3 + 5i\}.$$

Weryfikację wyniku przeprowadzamy za pomocą poniższego kodu. Warto podkreślić, że w zapisie y = imag(z) / (2*x) konieczny jest nawias w mianowniku, ponieważ w przeciwnym razie zmienna x znalazłaby się w liczniku:

```
z = -16 + 30*i
x = sqrt( (real(z) + abs(z)) / 2 )
y = imag(z) / (2*x)
w0 = x + y*i
print( 'Pierwiastek kwadratowy to zbiór:', {-w0, w0} )
Out: Pierwiastek kwadratowy to zbiór: {-5*I - 3, 5*I + 3}
```

Definicja. Niech liczba zespolona z będzie zapisana w postaci wykładniczej: $z = |z| \cdot e^{i\varphi}$. Pierwiastek stopnia n jest zbiorem n-elementowym:

$$\sqrt[n]{z} = \{\omega_0, \omega_1, \omega_2, \dots, \omega_{n-1}\},$$

gdzie poszczególne pierwiastki ω_k dla $k = 0, 1, 2, \dots, n-1$ obliczamy ze wzoru:

$$\omega_k = \sqrt[n]{|z|} \cdot e^{i\frac{\varphi + 2k\pi}{n}}. \tag{3.10}$$

Zauważmy, że dla dwóch kolejnych pierwiastków zachodzi równość:

$$\zeta := \frac{\omega_{k+1}}{\omega_k} = \frac{\sqrt[n]{|z|}\cdot e^{i\frac{\varphi+2(k+1)\pi}{n}}}{\sqrt[n]{|z|}\cdot e^{i\frac{\varphi+2k\pi}{n}}} = e^{i\frac{2\pi}{n}}, \quad \text{dla } k = 0, 1, 2, \dots, n-2.$$

Wniosek. Jeżeli znamy jeden z pierwiastków ω_0, to pozostałe możemy uzyskać, mnożąc kolejne pierwiastki przez liczbę $\zeta = e^{i\frac{2\pi}{n}}$, ponieważ:

$$\omega_{k+1} = \omega_k \cdot \zeta, \quad \text{dla } k = 0, 1, 2, \dots, n-2.$$

Wobec tego pierwiastki n-tego stopnia z liczby zespolonej z mają postać:

$$\sqrt[n]{z} = \{\omega_0, \omega_0 \cdot \zeta, \omega_0 \cdot \zeta^2, \dots, \omega_0 \cdot \zeta^{n-1}\}, \quad \text{gdzie } \zeta = e^{i\frac{2\pi}{n}}.$$

Przykład 3.21

Obliczymy pierwiastek trzeciego stopnia:

$$\sqrt[3]{(1+2i)^3}.$$

Zgadujemy, że jednym z pierwiastków jest liczba:

$$\omega_0 = 1 + 2i.$$

Mamy $n = 3$, więc liczba $\zeta = e^{i\frac{2\pi}{n}}$ na podstawie wzoru Eulera (3.6) jest równa:

$$\zeta = e^{i\frac{2\pi}{3}} = \cos\frac{2\pi}{3} + i\sin\frac{2\pi}{3} = -\frac{1}{2} + i\frac{\sqrt{3}}{2}.$$

Korzystając z powyższego wniosku, kolejne pierwiastki obliczamy jako:

$$\omega_1 = \omega_0 \cdot \zeta \quad \text{oraz} \quad \omega_2 = \omega_1 \cdot \zeta.$$

W **Sage** tę operację wykonujemy, korzystając z poniższych poleceń:

```
w0 = 1 + 2*i
n = 3
z = exp(i*2*pi/n)
pierwiastki = [expand(w0 * z^k) for k in range(n)]
```

```
pierwiastki
Out: [2*I + 1, (1/2*I - 1)*sqrt(3) - I - 1/2,
 -(1/2*I - 1)*sqrt(3) - I - 1/2]
```

Lista pierwiastki została utworzona za pomocą tzw. listy składanej z języka **Python**. Konstrukcja ta pozwala w zwięzły sposób wygenerować nową listę na podstawie innej (tutaj: range(n)), przy użyciu pętli **for** zapisanej w jednej linii. Stosujemy składnię:

[*element* **for** *zmienna* **in** *lista*].

Na przykład komenda

```
[x^2 for x in range(5)]
Out: [0, 1, 4, 9, 16]
```

pozwala uzyskać listę zawierającą kwadraty pięciu kolejnych liczb naturalnych, ponieważ polecenie range(n) tworzy ciąg kolejnych liczb od 0 do $n-1$. W powyższym kodzie zmienna x przechodzi po elementach listy [0, 1, 2, 3, 4], zaś zapis x^2 odpowiada za wyliczenie kwadratów kolejnych liczb.

Analogicznie w liście pierwiastki zmienna k przyjmuje wartości [0, 1, 2], a następnie obliczane są pierwiastki:

$$\omega_0 \cdot \zeta^k, \quad \text{dla } k = 0, 1, 2.$$

Funkcja **expand()** służy jedynie do uproszczenia wyników.

W dalszej części książki będziemy stosować powyższą metodę generowania pierwiastków, warto jednak wspomnieć o dwóch alternatywnych podejściach:

1. Bezpośrednie rozwiązanie równania $\omega_k^n = z$ za pomocą funkcji **solve()**:

```
var('z')
rozw = solve( (1+2*i)^3 == z^3, z )
```

Zwracane wyniki nie są zbyt czytelne, dlatego nie wyświetlamy ich w całości. Możemy jednak wykorzystać listę składaną, aby otrzymać przybliżenia numeryczne rozwiązań:

```
[r.rhs().n() for r in rozw]
Out: [1.00000000000000 + 2.00000000000000*I,
```

```
-2.23205080756888 - 0.133974596215562*I,
1.23205080756888 - 1.86602540378444*I]
```

Te same wyniki można uzyskać również za pomocą r.n() dla listy pierwiastki.

2. Wykorzystanie obiektu **CDF()** oraz funkcji **nth_root()** z argumentem all=True, która zwraca wszystkie pierwiastki danego stopnia:

```
CDF( (1+2*i)^3 ).nth_root(3, all=True)
Out: [1.2320508075688774 - 1.8660254037844386*I,
 1.0000000000000004 + 1.9999999999999998*I,
 -2.2320508075688776 - 0.1339745962155603*I]
```

Otrzymane wyniki nieznacznie się różnią ze względu na sposób reprezentacji liczb zmiennoprzecinkowych w języku **Python**.

3.3.2. Interpretacja geometryczna zbioru pierwiastków

Pierwiastki n-tego stopnia z liczby zespolonej $z \in \mathbb{C}$ tworzą na płaszczyźnie zespolonej wierzchołki n-kąta foremnego. Leżą one na okręgu o środku w punkcie $(0,0)$ i promieniu równym $\sqrt[n]{|z|}$. Każdy kolejny pierwiastek jest obrócony względem poprzedniego o kąt $\frac{2\pi}{n}$ wokół początku układu współrzędnych.

Przykład 3.22
Znajdziemy dokładne wartości elementów zbioru

$$\sqrt[4]{-7+24i}$$

oraz przedstawimy interpretację geometryczną uzyskanych pierwiastków.

Zapisujemy pierwiastek czwartego stopnia za pomocą dwóch pierwiastków kwadratowych:

$$\sqrt[4]{-7+24i} = \sqrt{\sqrt{-7+24i}}.$$

Teraz dwukrotnie stosujemy wzory (3.9) – najpierw dla pierwszego pierwiastka:

$$\omega_0^1 = x_1 + y_1 i: \quad x_1 = \sqrt{\frac{-7 + \sqrt{(-7)^2 + 24^2}}{2}} = 3, \quad y_1 = \frac{24}{2 \cdot 3} = 4$$

$$\sqrt{-7 + 24i} = \{-\omega_0^1, \omega_0^1\} = \{-3 - 4i, 3 + 4i\},$$

a następnie dla pierwiastka z jednego z uzyskanych wyników, na przykład $3 + 4i$:

$$\omega_0^2 = x_2 + y_2 i: \quad x_2 = \sqrt{\frac{3 + \sqrt{3^2 + 4^2}}{2}} = 2, \quad y_2 = \frac{4}{2 \cdot 2} = 1$$

$$\sqrt{3 + 4i} = \{-\omega_0^2, \omega_0^2\} = \{-2 - i, 2 + i\}.$$

W podobny sposób otrzymalibyśmy dwa pozostałe pierwiastki dla liczby $-3 - 4i$.

W **Sage** możemy iterować w pętli **for** po obiekcie range(2), który generuje listę [0, 1] – powoduje to, że wchodzimy do pętli dwukrotnie. Znak podkreślenia (_) zaznacza jedynie, że nie korzystamy ze zmiennej iteracyjnej (przyjmującej wartości odpowiednio 0 i 1). Po wyjściu z pętli przypisujemy wynik do zmiennej w0, która przechowuje dokładną wartość jednego z uzyskanych pierwiastków:

```
z = -7 + 24*i
for _ in range(2):
    x = sqrt( (real(z) + abs(z)) / 2 )
    y = imag(z) / (2*x)
    z = x + y*i
w0 = z
print('Jednym z pierwiastków jest liczba:')
show( LatexExpr(r'\omega_0='), w0 )
```

Out: Jednym z pierwiastków jest liczba:

$\omega_0 = i + 2$

Kolejne pierwiastki czwartego stopnia uzyskujemy poprzez pomnożenie liczby $\omega_0 := \omega_0^2$ przez czynnik $\zeta = e^{i\frac{2\pi}{n}}$, który dla $n = 4$ na podstawie wzoru Eulera (3.6) przyjmuje wartość:

$$\zeta = e^{i\frac{2\pi}{4}} = e^{i\frac{\pi}{2}} = \cos\frac{\pi}{2} + i\sin\frac{\pi}{2} = 0 + i \cdot 1 = i.$$

Wobec tego pozostałe pierwiastki czwartego stopnia równe są odpowiednio:

$$\omega_1 = \omega_0 \cdot i, \quad \omega_2 = \omega_1 \cdot i, \quad \omega_3 = \omega_2 \cdot i.$$

Oznacza to, że znając jeden pierwiastek czwartego stopnia, pozostałe możemy uzyskać poprzez mnożenie kolejnych pierwiastków przez jednostkę urojoną i:

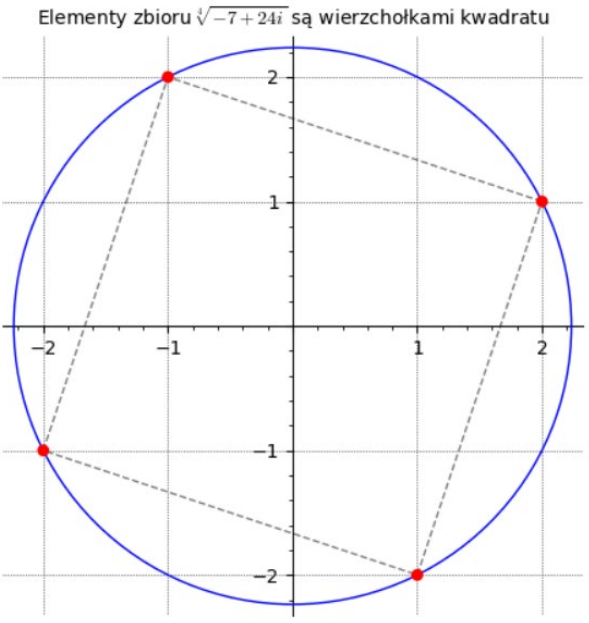
```
n = 4
z = exp(i*2*pi/n)
pierwiastki = [w0 * z^k for k in range(n)]
pierwiastki
Out: [I + 2, 2*I - 1, -I - 2, -2*I + 1]
```

Zgodnie z przedstawioną wcześniej interpretacją geometryczną liczby te tworzą wierzchołki wielokąta foremnego, przy czym dla $n = 4$ jest to kwadrat. Rzeczywiście mnożenie kolejnych pierwiastków przez i powoduje obrót o kąt 90° (por. przykład 3.8). Poniższy kod wizualizuje wynik dla pierwiastka dowolnego stopnia – wystarczy umieścić rozwiązania w liście o nazwie pierwiastki:

```
rys = points( pierwiastki, color='red', size=40, zorder=7,
 title=r'Elementy zbioru $\sqrt[4]{-7+24i}$ są wierzchołkami kwadratu' )
rys += circle( (0, 0), abs(w0), aspect_ratio=1 )
rys += polygon( pierwiastki, color='gray', linestyle='--',
 fill=False, gridlines=True )
rys
```

Efekt uruchomienia komórki z tym kodem przedstawiono na rysunku 3.20. Przeanalizujemy sposób działania poszczególnych poleceń:

- funkcja **points()** zaznacza na rysunku punkty zawarte w liście pierwiastki kolorem czerwonym – tworzą one wierzchołki kwadratu;
- funkcja **circle()** kreśli niebieski okrąg o środku w początku układu współrzędnych i promieniu równym $|\omega_0|$, natomiast argument aspect_ratio określa stosunek jednostek na pionowej osi OY do jednostek na osi poziomej OX – ustawiamy jego wartość na 1, aby na obu osiach jednostki miały taką samą długość;
- funkcja **polygon()** rysuje wielokąt szarą linią przerywaną, argument fill=False sprawia, że figura jest pusta w środku, natomiast komenda gridlines=True dodaje linie siatki do rysunku – możemy ją umieścić jako argument dowolnej funkcji.



Rysunek 3.20. Interpretacja geometryczna pierwiastka czwartego stopnia

Przy okazji warto zauważyć, że składnia wykorzystująca czynnik ζ powoduje, że pierwiastki są umieszczane w liście pierwiastki w kolejności przeciwnej do ruchu wskazówek zegara, dzięki czemu funkcja **polygon()** poprawnie rysuje kwadrat.

Przykład 3.23

Wyznaczymy dokładne wartości elementów zbioru

$$\sqrt[6]{1+i}$$

oraz zaznaczymy uzyskane liczby na płaszczyźnie zespolonej.

Jeden z pierwiastków liczby $z = 1 + i$ możemy obliczyć na podstawie wzoru (3.10), mianowicie jako:

$$\omega_0 = \sqrt[n]{|z|} \cdot e^{i\frac{\varphi}{n}}, \tag{3.11}$$

dla $n = 6$. Wartości modułu $|z| = \sqrt{2}$ i argumentu $\varphi = \frac{\pi}{4}$ liczby z zostały wyznaczone wcześniej w przykładzie 3.10. Po przekształceniach otrzymujemy:

$$\omega_0 = \sqrt[6]{|z|} \cdot e^{i\frac{\varphi}{6}} = \sqrt[6]{\sqrt{2}} \cdot e^{i\frac{\frac{\pi}{4}}{6}} = \sqrt[12]{2} \cdot e^{i\frac{\pi}{24}}.$$

Aby uzyskać pozostałe rozwiązania, mnożymy kolejne pierwiastki przez czynnik

$$\zeta = e^{i\frac{2\pi}{n}} = e^{i\frac{2\pi}{6}} = e^{i\frac{\pi}{3}} = \cos\frac{\pi}{3} + i\sin\frac{\pi}{3} = \frac{1}{2} + i\frac{\sqrt{3}}{2}.$$

Ostatecznie szukane pierwiastki przyjmują postać:

$$\omega_1 = \omega_0 \cdot \zeta^1, \quad \omega_2 = \omega_0 \cdot \zeta^2, \quad \omega_3 = \omega_0 \cdot \zeta^3,$$
$$\omega_4 = \omega_0 \cdot \zeta^4, \quad \omega_5 = \omega_0 \cdot \zeta^5.$$

W **Sage** wyliczamy potrzebne wielkości oraz stosujemy wzór (3.11). Pierwiastek rzeczywisty stopnia n możemy zapisać równoważnie za pomocą potęgowania:

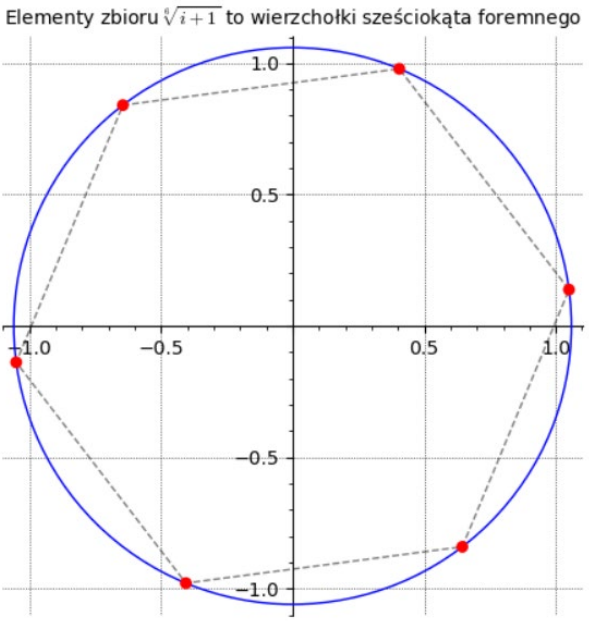
```
z = i + 1
n = 6
modul = abs(z)
phi = arg(z)
w0 = modul^(1/n) * exp(i*phi/n)
pierwiastki = [w0 * exp(i*2*k*pi/n) for k in range(n)]
pierwiastki
```

Uzyskane pierwiastki nie są czytelne, dlatego tutaj je pomijamy. W kodzie zastosowaliśmy listę składaną, która generuje elementy na podstawie formuły:

$$\omega_k = \omega_0 \cdot e^{i\frac{2k\pi}{n}} = \omega_0 \cdot \zeta^k, \quad \text{dla } k = 0, 1, 2, 3, 4, 5.$$

wynikającej ze wzoru (3.10). Następnie wynik przypisujemy do zmiennej pierwiastki. Elementy tej listy zaznaczamy na płaszczyźnie zespolonej, stosując kod z przykładu 3.22:

```
rys = points( pierwiastki, color='red', size=40, zorder=7,
 title=r'Elementy zbioru $\sqrt[6]{i+1}$ ' +
 'to wierzchołki sześciokąta foremnego' )
rys += circle( (0, 0), abs(w0), aspect_ratio=1 )
rys += polygon( pierwiastki, color='gray', linestyle='--',
 fill=False, gridlines=True )
rys
```



Rysunek 3.21. Interpretacja geometryczna pierwiastka szóstego stopnia

Na rysunku 3.21 przedstawiono uzyskany rezultat. Pierwiastek szóstego stopnia jest zbiorem sześcioelementowym, więc jego elementy wyznaczają na płaszczyźnie wierzchołki sześciokąta foremnego. Ponieważ wszystkie pierwiastki mają jednakowy moduł, otrzymana figura jest wpisana w okrąg o środku w początku układu współrzędnych.

3.3.3. Wielokąty foremne

Poznane własności pierwiastków z liczb zespolonych na płaszczyźnie zespolonej możemy wykorzystać do konstruowania wielokątów foremnych w układzie współrzędnych.

Przykład 3.24
Narysujemy siedmiokąt foremny, którego jeden z wierzchołków leży w punkcie $(3, 1)$.

Przyjmujemy, że jednym z elementów pewnego pierwiastka siódmego stopnia jest liczba zespolona odpowiadająca punktowi $(3, 1)$, czyli:

$$\omega_0 = 3 + i.$$

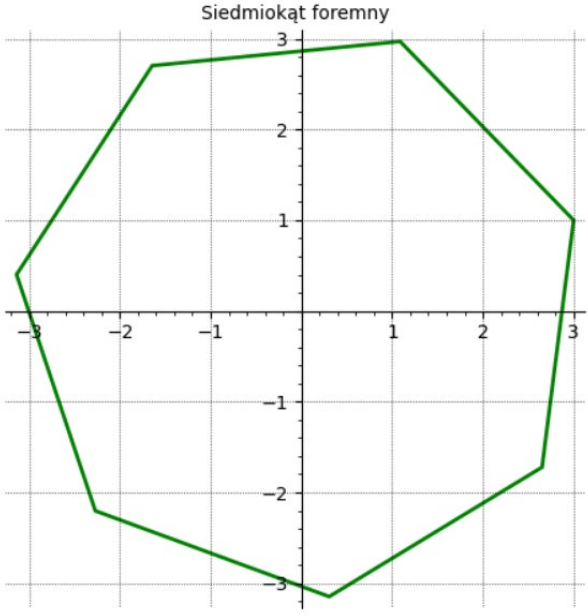
Pozostałe pierwiastki uzyskujemy, mnożąc kolejne elementy przez czynnik

$$\zeta = e^{i\frac{2\pi}{7}}.$$

Szukany siedmiokąt foremny możemy zatem uzyskać za pomocą funkcji **polygon()**, bazując na kodzie z przykładu 3.23:

```
w0 = 3 + i
n = 7


pierwiastki = [w0 * exp(i*2*k*pi/n) for k in range(n)]
polygon( pierwiastki, color='green', thickness=2, aspect_ratio=1,
 fill=False, gridlines=True, title='Siedmiokąt foremny' )
```

Otrzymana figura przedstawiona jest na rysunku 3.22. W liście pierwiastki znajdują się wszystkie pierwiastki siódmego stopnia. Pierwszym elementem jest liczba w0, natomiast kolejne otrzymujemy po przemnożeniu przez czynnik ζ. Poszczególne pierwiastki są generowane w kolejności przeciwnej do ruchu wskazówek zegara.



Rysunek 3.22. Siedmiokąt foremny o wierzchołku w punkcie $(3, 1)$

Przykład 3.25
Narysujemy sześciokąt foremny $\mathcal{F}$, którego jednym z wierzchołków jest punkt $A = (4, 2)$, a środkiem symetrii – punkt $S = (1, 2)$.

Wykonujemy przesunięcie równoległe punktów A i S o jedną jednostkę w lewo i dwie jednostki w dół, tak aby środkiem symetrii figury $\mathcal{F}'$ (uzyskanej w wyniku tego przekształcenia) był początek układu współrzędnych $S' = (0, 0)$. W wyniku tej operacji otrzymujemy współrzędne jednego z wierzchołków przesuniętej figury:

$$A' = (3, 0).$$

Następnie, analogicznie jak w przykładzie 3.24, przyjmujemy, że punktowi A' odpowiada liczba zespolona

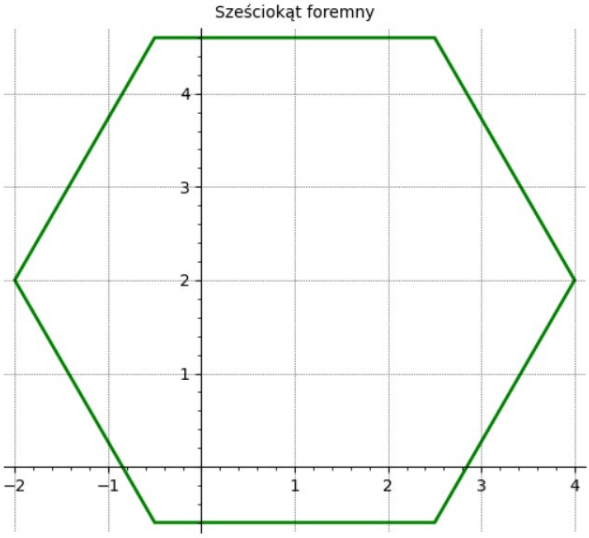
$$\omega_0 = 3,$$

która jest jednym z elementów pewnego pierwiastka szóstego stopnia. Kolejne wierzchołki figury $\mathcal{F}'$ otrzymujemy, mnożąc przez czynnik

$$\zeta = e^{i\frac{2\pi}{6}}.$$

Po wyznaczeniu wszystkich wierzchołków figury $\mathcal{F}'$ wykonujemy transformację odwrotną – przesuwamy z powrotem wszystkie wierzchołki równolegle o wektor $[1, 2]$. W zapisie zespolonym oznacza to dodanie liczby $1 + 2i$ do każdego pierwiastka tworzącego wierzchołki figury $\mathcal{F}'$. Czynności te wykonujemy w **Sage** za pomocą poniższego kodu:

```
A = 4 + 2*i
S = 1 + 2*i

w0 = A - S
n = 6

pierwiastki = [w0 * exp(i*2*k*pi/n) + S for k in range(n)]
polygon( pierwiastki, color='green', thickness=2, aspect_ratio=1,
 fill=False, gridlines=True, title='Sześciokąt foremny' )
```



Rysunek 3.23. Sześciokąt foremny o wierzchołku w punkcie $(4, 2)$ i środku symetrii $(1, 2)$

Wynik prezentujemy na rysunku 3.23. Definiujemy zmienne `A` i `S` jako liczby zespolone odpowiadające punktom A i S. Zmienna `w0` reprezentuje punkt A' uzyskany po przesunięciu równoległym. W liście `pierwiastki` wyznaczane są wierzchołki figury $\mathcal{F}'$, natomiast dodając zmienną `S` (zapis `+ S`), wykonujemy transformację odwrotną, czyli przesunięcie wszystkich wierzchołków do położenia początkowego.

Przykład 3.26

Narysujemy ciąg kwadratów $(K_1, K_2, K_3, \ldots, K_7)$, którego pierwszym elementem jest kwadrat K_1 o wierzchołku w punkcie $(1, 3)$, a każdy kolejny kwadrat jest wpisany w poprzedni, tak że jego wierzchołki stanowią środki boków poprzedniego kwadratu.

Weźmy kwadrat K_2 wpisany w kwadrat K_1. Wyznaczymy stosunek σ odległości wierzchołków figur K_2 i K_1 od początku układu współrzędnych. Wartość ta jest równa stosunkowi promienia okręgu wpisanego r do promienia okręgu opisanego R na kwadracie K_1, czyli:

$$\sigma = \frac{r}{R} = \frac{\frac{1}{2}a}{\frac{\sqrt{2}}{2}a} = \frac{\sqrt{2}}{2}.$$

Widzimy, że wynik nie zależy od długości boku a kwadratu K_1. Następnie określamy miarę kąta θ, o który przesunięte są wierzchołki figury K_2 względem kwadratu K_1:

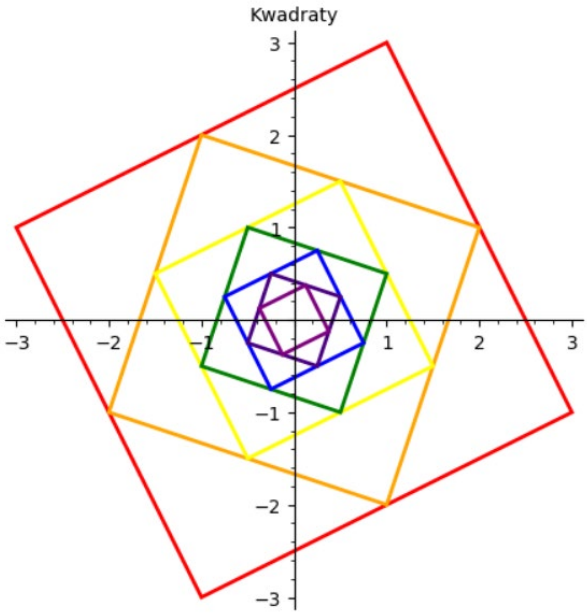
$$\theta = \frac{\pi}{4}.$$

Wzory te są prawdziwe dla dowolnych dwóch kolejnych kwadratów. Wobec tego dla liczby $\omega_0 = 1 + 3i$ współrzędne wierzchołków j-tego kwadratu (wyrażone za pomocą liczb zespolonych), gdzie $j = 0, 1, 2, \ldots, 6$, uzyskujemy jako:

$$\omega_k^j = \omega_0 \cdot e^{i\frac{2k\pi}{n}} \cdot e^{ij\theta} \cdot \sigma^j, \quad \text{dla } k = 0, 1, 2, 3.$$

Czynnik $e^{ij\theta}$ powoduje obrót wierzchołków kolejnych kwadratów o kąt θ, natomiast czynnik σ^j odpowiada za ich skalowanie. W **Sage** stosujemy poniższe polecenia:

```
w0 = 1 + 3*i
n = 4
sigma = sqrt(2)/2
theta = pi/4

rys = Graphics()
kolory = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'purple']

for j in range(7):
    pierwiastki = [w0 * exp(i*2*k*pi/n) *
    exp(i*j*theta) * sigma^j for k in range(n)]
    rys += polygon( pierwiastki, color=kolory[j],
    thickness=2, aspect_ratio=1,
    fill=False, title='Kwadraty' )
rys
```



Rysunek 3.24. Ciąg kwadratów wpisanych

Efekt działania kodu przedstawiono na rysunku 3.24. Definiujemy zmienną rys jako pusty obiekt graficzny Graphics() oraz określamy kolory poszczególnych figur w liście kolory. Następnie w pętli **for** znajdujemy wierzchołki kolejnych kwadratów oraz dodajemy wielokąty uzyskane za pomocą funkcji **polygon()** do obiektu rys, korzystając z operatora +=. Przedstawione w tym rozdziale fragmenty kodu można łatwo dostosować do innych danych – wystarczy zmodyfikować położenie punktu początkowego.

3.4. Równania w ciele liczb zespolonych

W ciele liczb zespolonych każde równanie wielomianowe n-tego stopnia ma dokładnie n pierwiastków zespolonych, z uwzględnieniem ich krotności. Fakt ten wynika z tzw. podstawowego twierdzenia algebry, zgodnie z którym każdy niezerowy wielomian o współczynnikach zespolonych ma co najmniej jeden pierwiastek zespolony. Z tego powodu mówimy, że ciało liczb zespolonych jest algebraicznie domknięte.

Własność tę można wyraźnie zobrazować dla równań kwadratowych. W zbiorze liczb rzeczywistych liczba rozwiązań takiego równania zależy od znaku wyróżnika Δ. W szczególności, gdy $\Delta < 0$, brak jest rozwiązań rzeczywistych. W zbiorze liczb zespolonych każde równanie kwadratowe ma natomiast dokładnie

dwa rozwiązania (licząc z krotnościami). Dzieje się tak nawet wtedy, gdy wyróżnik jest ujemny – wówczas rozwiązaniami są liczby zespolone zawierające jednostkę urojoną i. Co więcej, w ciele liczb zespolonych nadal można stosować znane wzory na pierwiastki równania kwadratowego, co rozszerza zakres ich zastosowań poza zbiór liczb rzeczywistych.

3.4.1. Równość liczb zespolonych

Dwie liczby zespolone zapisane w postaci algebraicznej, czyli:

$$z_1 = x_1 + y_1 i \quad \text{oraz} \quad z_2 = x_2 + y_2 i, \quad \text{gdzie } x_1, y_1, x_2, y_2 \in \mathbb{R},$$

są równe tylko wtedy, gdy równe są ich części rzeczywiste i części urojone, tzn.:

$$\begin{cases} x_1 = x_2 \\ y_1 = y_2. \end{cases}$$

Przykład 3.27

Znajdziemy liczby rzeczywiste $x, y \in \mathbb{R}$ spełniające równanie:

$$(1 + ix)(y - 2i) = 3 - 4i.$$

Wykonujemy mnożenie wyrażeń w nawiasach po lewej stronie równania:

$$y - 2i + xyi + 2x = 3 - 4i.$$

Wyodrębniamy po obu stronach części rzeczywistą i urojoną:

$$(2x + y) + (xy - 2)i = 3 - 4i.$$

Stosujemy warunek równości liczb zespolonych w postaci algebraicznej i otrzymujemy układ dwóch równań rzeczywistych:

$$\begin{cases} 2x + y = \ \ 3 \\ xy - 2 = -4. \end{cases}$$

Z pierwszego równania wyznaczamy zmienną $y = 3 - 2x$ i podstawiamy ją do drugiego równania, uzyskując równanie kwadratowe:

$$x(3 - 2x) - 2 = -4 \ \rightarrow \ 3x - 2x^2 - 2 = -4 \ \rightarrow \ 2x^2 - 3x - 2 = 0.$$

Liczymy wyróżnik

$$\Delta = (-3)^2 - 4 \cdot 2 \cdot (-2) = 25$$

oraz zapisujemy rozwiązania równania:

$$x_1 = \frac{3-5}{2\cdot 2} = -\frac{1}{2} \quad \vee \quad x_2 = \frac{3+5}{2\cdot 2} = 2.$$

Wracamy do wyznaczonej zmiennej $y = 3 - 2x$ i podajemy rozwiązania układu równań:

$$\begin{cases} x = -\frac{1}{2} \\ y = 4 \end{cases} \quad \vee \quad \begin{cases} x = 2 \\ y = -1. \end{cases}$$

W **Sage** układy równań rozwiązujemy za pomocą funkcji **solve()**. Jako pierwszy argument przekazujemy listę, w której porównujemy części rzeczywiste oraz urojone lewej i prawej strony równania:

```
var('x y')
rownanie = (1 + i*x) * (y - 2*i) == 3 - 4*i
solve( [rownanie.lhs().real() == rownanie.rhs().real(),
 rownanie.lhs().imag() == rownanie.rhs().imag()], [x, y] )
Out: [[x == 2, y == -1], [x == (-1/2), y == 4]]
```

Dwie liczby zespolone zapisane w postaci wykładniczej (lub trygonometrycznej), czyli:

$$z_1 = |z_1| \cdot e^{i\varphi_1} \quad \text{oraz} \quad z_2 = |z_2| \cdot e^{i\varphi_2}.$$

są równe tylko wtedy, gdy równe są ich moduły, natomiast argumenty różnią się o wielokrotność kąta 2π, tzn.:

$$\begin{cases} |z_1| = |z_2| \\ \varphi_1 = \varphi_2 + 2k\pi, \end{cases} \quad \text{dla pewnej liczby } k \in \mathbb{Z}.$$

Przykład 3.28

W liczbach zespolonych rozwiążemy równanie:

$$|z|^3 = i \cdot (\overline{z})^5.$$

Stosujemy postać wykładniczą liczby $z = |z| \cdot e^{i\varphi}$ i zapisujemy równanie w postaci:

$$|z|^3 = i \cdot \left(|z| \cdot e^{-i\varphi}\right)^5 \ \rightarrow \ |z|^3 = i \cdot |z|^5 \cdot e^{-5i\varphi}.$$

Przenosimy wszystkie wyrazy na jedną stronę i wyłączamy czynnik $|z|^3$ przed nawias:

$$|z|^5 \cdot i \cdot e^{-5i\varphi} - |z|^3 = 0 \ \rightarrow \ |z|^3 \cdot \left(|z|^2 \cdot i \ \cdot e^{-5i\varphi} - 1\right) = 0.$$

Otrzymujemy dwa przypadki:

$$|z|^3 = 0 \quad \vee \quad |z|^2 \cdot i \cdot e^{-5i\varphi} - 1 = 0.$$

Pierwsza równość spełniona jest tylko wtedy, gdy $|z| = 0$. Ponieważ moduł reprezentuje odległość liczby zespolonej od początku układu współrzędnych, warunek ten jest prawdziwy tylko dla liczby $z_0 = 0$.

W drugim przypadku przenosimy liczbę 1 na prawą stronę równania i zapisujemy liczby i oraz 1 w postaci wykładniczej (3.4):

$$|z|^2 \cdot e^{i\frac{\pi}{2}} \cdot e^{-5i\varphi} = 1 \ \rightarrow \ |z|^2 \cdot e^{i\left(\frac{\pi}{2} - 5\varphi\right)} = 1 \cdot e^{i0}.$$

Stosujemy warunek na równość liczb zespolonych w postaci wykładniczej:

$$\begin{cases} |z|^2 = 1 \\ \dfrac{\pi}{2} - 5\varphi = 0 + 2k\pi \end{cases} (k \in \mathbb{Z}).$$

Moduł jest liczbą nieujemną, więc pierwszą równość można zapisać jako $|z| = 1$. Oznacza to, że rozwiązania leżą na okręgu jednostkowym. Z drugiego równania wyznaczamy zaś argument jako:

$$\varphi = \frac{\pi}{10} + \frac{2}{5}k\pi, \quad \text{gdzie } k \in \mathbb{Z}.$$

Stosujemy konwencję z **Sage**, zgodnie z którą argument główny liczby zespolonej należy do przedziału $(-\pi, \pi]$. Dzieje się tak dla parametrów $k \in \mathbb{Z}$ ze zbioru $\{-2, -1, 0, 1, 2\}$. Otrzymujemy zatem pięć kolejnych rozwiązań zespolonych w postaci:

$$z_1 = e^{-i\frac{7\pi}{10}} \ \vee \ z_2 = e^{-i\frac{3\pi}{10}} \ \vee \ z_3 = e^{i\frac{\pi}{10}} \ \vee \ z_4 = e^{i\frac{5\pi}{10}} \ \vee \ z_5 = e^{i\frac{9\pi}{10}}.$$

W celu rozwiązania tego równania w **Sage**, zapisujemy moduł i sprzężenie w początkowym równaniu jako $|z| = \sqrt{x^2 + y^2}$ oraz $\overline{z} = x - yi$. Pierwsze rozwiązanie odpowiada liczbie $z_0 = 0$, drugie jest równe $z_4 = i$, natomiast trzecie rozwiązanie, [x == 0, y == -I], odrzucamy, ponieważ wartość zmiennej y == -I nie jest liczbą rzeczywistą. Pozostałe wyniki zostały zapisane w postaci algebraicznej po zastosowaniu wzoru Eulera (3.6):

```
rownanie = (sqrt(x^2+y^2))^3 == i * (x-y*i)^5
solve( [rownanie.lhs().real() == rownanie.rhs().real(),
 rownanie.lhs().imag() == rownanie.rhs().imag()], [x, y] )
Out: [[x == 0, y == 0],
 [x == 0, y == 1],
 [x == 0, y == -I],
 [x == -1/2*sqrt(-1/2*sqrt(5) + 5/2),
 y == -1/4*sqrt(5) - 1/4],
 [x == 1/2*sqrt(-1/2*sqrt(5) + 5/2),
 y == -1/4*sqrt(5) - 1/4],
 [x == -1/2*sqrt(1/2*sqrt(5) + 5/2),
 y == 1/4*sqrt(5) - 1/4],
 [x == 1/2*sqrt(1/2*sqrt(5) + 5/2),
 y == 1/4*sqrt(5) - 1/4]]
```

3.4.2. Równania wielomianowe

Równanie kwadratowe postaci $az^2 + bz + c = 0$, gdzie $a, b, c \in \mathbb{C}$ oraz $a \neq 0$, ma dwa pierwiastki zespolone:

$$z_1 = \frac{-b-\delta}{2a} \quad \vee \quad z_2 = \frac{-b+\delta}{2a}, \tag{3.12}$$

przy czym liczba $\delta \in \mathbb{C}$ jest dowolnym elementem zbioru $\sqrt{\Delta}$, gdzie $\Delta = b^2 - 4ac$.

Przykład 3.29

W ciele liczb zespolonych rozwiążemy równanie kwadratowe:

$$z^2 - (1 - i)z + 2 - 2i = 0.$$

Obliczamy wyróżnik trójmianu kwadratowego:

$$\Delta = (1-i)^2 - 4 \cdot 1 \cdot (2-2i) = 1 - 2i - 1 - 8 + 8i = -8 + 6i.$$

Chcemy wyznaczyć jeden z elementów zbioru:

$$\sqrt{\Delta} = \sqrt{-8+6i} = \{-\delta, \delta\}.$$

Korzystając ze wzorów (3.9), znajdujemy pierwiastek $\delta = x + yi$, gdzie:

$$x = \sqrt{\frac{-8 + \sqrt{(-8)^2 + 6^2}}{2}} = 1, \qquad y = \frac{6}{2 \cdot 1} = 3.$$

Zatem $\delta = 1 + 3i$. Teraz stosujemy wzory (3.12), aby wyznaczyć pierwiastki równania kwadratowego:

$$z_1 = \frac{1 - i - (1+3i)}{2 \cdot 1} = -2i \quad \vee \quad z_2 = \frac{1 - i + 1 + 3i}{2 \cdot 1} = 1 + i.$$

Poniżej przedstawiamy przykładowy kod, który rozwiązuje ten problem w **Sage**:

```
var('x y')
z = x + y*i
rownanie = z^2 - (1-i)*z + 2-2*i
solve( [rownanie.real(), rownanie.imag()], [x, y] )
Out: [[x == 1, y == 1],
 [x == (-3/2*I + 1/2), y == (1/2*I - 1/2)],
 [x == (3/2*I + 1/2), y == (-1/2*I - 1/2)],
 [x == 0, y == -2]]
```

Tym razem zmienna `rownanie` zawiera wyrażenie, a nie równość, dlatego nie korzystamy z funkcji **lhs()** i **rhs()**. W funkcji **solve()** nie musimy jawnie zapisywać równości, ponieważ domyślnie przyjmuje ona, że prawa strona równania jest równa 0. Tutaj zapis `solve(rownanie, z)` nie zwróciłby oczekiwanego wyniku – pierwiastek kwadratowy z liczby zespolonej nie zostałby obliczony, a wynik pojawiłby się w symbolicznej postaci `sqrt(6*I - 8)`. W naszym kodzie używamy postaci algebraicznej liczby zespolonej $z = x + yi$, gdzie $x, y \in \mathbb{R}$, dlatego drugie i trzecie rozwiązanie odrzucamy jako nienależące do zbioru liczb rzeczywistych.

Przykład 3.30
Rozwiążemy równanie trzeciego stopnia:

$$z^3 - iz^2 + z - i = 0.$$

Stosujemy metodę grupowania:

$$z^2(z-i) + (z-i) = 0 \;\rightarrow\; (z-i)(z^2+1) = 0.$$

Korzystając z równości $i^2 = -1$, rozkładamy wielomian drugiego stopnia na czynniki liniowe:

$$z^2 + 1 = (z-i)(z+i).$$

Następnie z równania otrzymujemy pierwiastki:

$$(z-i)^2(z+i) = 0 \;\rightarrow\; z = i \;\vee\; z = -i.$$

Rozwiązanie $z = i$ jest pierwiastkiem dwukrotnym, natomiast $z = -i$ – jednokrotnym.

W ciele liczb zespolonych suma krotności pierwiastków jest zawsze równa stopniowi równania. W **Sage** stosujemy prostszą składnię:

```
var('z')
rownanie = z^3 - i*z^2 + z - i
solve(rownanie, z)
Out: [z == I, z == -I]
```

W celu określenia krotności pierwiastków można wykorzystać funkcję **roots()**:

```
rownanie.roots()
Out: [(I, 2), (-I, 1)]
```

Jeżeli wszystkie współczynniki wielomianu w równaniu są liczbami rzeczywistymi, to zachodzi następująca zależność:

z_0 jest pierwiastkiem o krotności $k \iff \overline{z_0}$ jest pierwiastkiem o krotności k.

Zatem jeżeli liczba zespolona $z_0 \in \mathbb{C}$ jest rozwiązaniem równania, to jej sprzężenie również.

Przykład 3.31
Rozwiążemy równanie o współczynnikach rzeczywistych:

$$z^4 + 13z^2 + 36 = 0.$$

Wprowadzamy zmienną pomocniczą $z^2 = t$ i rozwiązujemy uzyskane równanie kwadratowe:

$$t^2 + 13t + 36 = 0.$$

Liczymy wyróżnik $\Delta = 13^2 - 4 \cdot 1 \cdot 36 = 25$ oraz znajdujemy pierwiastki tego równania:

$$t_1 = \frac{-13-5}{2 \cdot 1} = -9 \quad \vee \quad t_2 = \frac{-13+5}{2 \cdot 1} = -4.$$

Wracamy do podstawienia i otrzymujemy dwa równania zespolone:

$$z^2 = -9 \quad \vee \quad z^2 = -4.$$

Ostateczne rozwiązania to:

$$z = -3i \ \vee \ z = 3i \ \vee \ z = -2i \ \vee \ z = 2i.$$

Faktycznie widzimy, że występują one w parach sprzężonych. W **Sage** stosujemy kod:

```
rownanie = z^4 + 13*z^2 + 36
solve(rownanie, z)
Out: [z == (-2*I), z == (2*I), z == (-3*I), z == (3*I)]
```

3.4.3. Twierdzenie Bézouta

Twierdzenie (Bézouta).
Liczba $z_0 \in \mathbb{C}$ jest pierwiastkiem równania $W(z) = 0$ tylko wtedy, gdy $(z - z_0) \mid W(z)$.

Oznacza to, że liczba zespolona z_0 jest rozwiązaniem równania $W(z) = 0$ tylko wtedy, gdy wielomian W jest podzielny przez dwumian $(z - z_0)$. Innymi słowy, dwumian $(z - z_0)$ pojawia się w rozkładzie wielomianu W na czynniki liniowe.

Przykład 3.32
W ciele liczb zespolonych rozwiążemy równanie wielomianowe:

$$z^5 - z^4 + 13z^3 - 13z^2 + 36z - 36 = 0.$$

Jeżeli wszystkie współczynniki wielomianu W w równaniu $W(z) = 0$ są liczbami całkowitymi, to ewentualne całkowite rozwiązania muszą być dzielnikami wyrazu wolnego, czyli w tym przypadku liczby –36. Odgadujemy, że liczba $z_0 = 1$ jest pierwiastkiem tego równania, a następnie stosujemy schemat Hornera, aby zmniejszyć stopień wielomianu W:

	1	−1	13	−13	36	−36
$z_0 = 1$	↓	$1 \cdot 1 = 1$	$1 \cdot 0 = 0$	$1 \cdot 13 = 13$	$1 \cdot 0 = 0$	$1 \cdot 36 = 36$
	1	0	13	0	36	0

Liczby w pierwszym wierszu są współczynnikami wielomianu W. W kolejnych krokach wykonujemy mnożenie liczby z dolnego wiersza przez liczbę z_0 (zaznaczoną kolorem czerwonym) oraz dodawanie wartości uzyskanych w dwóch pierwszych wierszach. Liczby na dole są współczynnikami ilorazu będącego wynikiem dzielenia wielomianu W przez dwumian $(z - z_0)$. W prawym dolnym rogu tabeli uzyskujemy resztę z dzielenia, która w przypadku dzielenia przez dwumian $(z - z_0)$ zawsze jest równa $W(z_0)$.

Otrzymujemy rozkład wielomianu W na czynniki:

$$W(z) = (z - 1)(z^4 + 13z^2 + 36).$$

Od tej pory postępujemy analogicznie jak w przykładzie 3.31. W **Sage** możemy skorzystać z funkcji **coefficients()**, która zwraca współczynniki wielomianu wraz z odpowiadającymi im potęgami zmiennej:

```
W(z) = z^5 - z^4 + 13*z^3 - 13*z^2 + 36*z - 36
W(z).coefficients()
Out: [[-36, 0], [36, 1], [-13, 2], [13, 3], [-1, 4], [1, 5]]
```

Wartość konkretnego współczynnika można też uzyskać za pomocą funkcji **coefficient()**. Poniżej wyznaczamy współczynnik stojący przy zmiennej z^2:

```
W(z).coefficient(z^2)
Out: -13
```

Przykład 3.33

Wiedząc, że jednym z rozwiązań równania

$$z^4 - 2z^3 + 6z^2 - 8z + 8 = 0$$

jest liczba $z_0 = 1 - i$, wyznaczymy pozostałe pierwiastki tego równania.

Wykonujemy dzielenie wielomianu W, występującego po lewej stronie równania, przez dwumian $(z - z_0)$ za pomocą schematu Hornera:

	1	-2	6	-8	8
$z_0 = 1 - i$	↓	$1 - i$	-2	$4 - 4i$	-8
	1	$-1 - i$	4	$-4 - 4i$	0

Otrzymujemy następujący rozkład na czynniki:

$$W(z) = \big(z - (1 - i)\big)(z^3 + (-1 - i)z^2 + 4z - 4 - 4i).$$

Wszystkie współczynniki wielomianu W są rzeczywiste, zatem rozwiązaniem równania jest również sprzężenie $\overline{z_0} = 1 + i$. Tym razem korzystamy ze schematu Hornera, aby wykonać dzielenie przez dwumian $(z - \overline{z_0})$:

	1	$-1 - i$	4	$-4 - 4i$
$z_0 = 1 + i$	↓	$1 + i$	0	$4 + 4i$
	1	0	4	0

W efekcie otrzymujemy równanie:

$$\big(z - (1 - i)\big)\big(z - (1 + i)\big)(z^2 + 4) = 0.$$

Z uzyskanej postaci iloczynowej możemy odczytać rozwiązania:

$$z = 1 - i \ \vee \ z = 1 + i \ \vee \ z = -2i \ \vee \ z = 2i.$$

W **Sage** równania do czwartego stopnia o współczynnikach całkowitych możemy rozwiązać bezpośrednio za pomocą funkcji **solve()**:

```
var('z')
rownanie = z^4 - 2*z^3 + 6*z^2 - 8*z + 8
solve(rownanie, z)
Out: [z == (-2*I), z == (2*I), z == (-I + 1), z == (I + 1)]
```

Więcej informacji na temat liczb zespolonych oraz ich zastosowań można znaleźć w pozycjach [2] i [3].

Dużo uśmiechu! 😁

4. Zadania

4.1. Zadania z rozwiązaniami

Zadanie 4.1

Dana jest liczba zespolona:

$$z = -1 + i\sqrt{3}.$$

a) Znajdź moduł i argument główny liczby z.

b) Zapisz liczbę z w postaci wykładniczej.

c) Wyznacz liczbę zespoloną uzyskaną po obrocie liczby z na płaszczyźnie zespolonej o kąt $\theta = \frac{\pi}{4}$ przeciwnie do ruchu wskazówek zegara. Wynik przedstaw w postaci algebraicznej.

d) Przedstaw obrót liczby z o kąt $\theta = 75°$ zgodnie z ruchem wskazówek zegara.

a) Aby obliczyć moduł liczby z, korzystamy ze wzoru (3.3):

$$|z| = \sqrt{(-1)^2 + \left(\sqrt{3}\right)^2} = 2.$$

Argument można odczytać z rysunku – na płaszczyźnie zespolonej punkty $(0,0), \left(-1,\sqrt{3}\right)$ i $\left(0,\sqrt{3}\right)$ tworzą trójkąt o kątach 30°, 60° i 90°. Ponieważ liczba z leży w drugiej ćwiartce, jej argument główny jest równy:

$$\varphi = \frac{\pi}{2} + \frac{\pi}{6} = \frac{2\pi}{3}.$$

W **Sage** wynik ten uzyskujemy za pomocą następującego kodu:

```
z = -1 + i*sqrt(3)
modul, phi = abs(z).simplify(), arg(z).simplify()
print('Moduł liczby zespolonej:')
show( LatexExpr('|z|='), modul )
```

```
print('Argument liczby zespolonej:')
show( LatexExpr(r'\varphi='), phi )
```
Out: Moduł liczby zespolonej:

$|z| = 2$

Argument liczby zespolonej:

$\varphi = \frac{2}{3}\pi$

b) Znając moduł i argument liczby z, zapisujemy ją w postaci wykładniczej, czyli:

$$z = |z| \cdot e^{i\varphi} = 2 \cdot e^{i\frac{2\pi}{3}}.$$

Stosujemy argument hold=True, aby wartość funkcji **exp()** nie została wyliczona:

```
postac_wykladnicza = modul * exp(i*phi, hold=True)
print('Postać wykładnicza:')
show( LatexExpr('z='), postac_wykladnicza )
```
Out: Postać wykładnicza:

$z = 2e^{\left(\frac{2}{3}i\pi\right)}$

c) Aby znaleźć liczbę zespoloną z', uzyskaną po obrocie liczby z o kąt θ przeciwnie do ruchu wskazówek zegara, stosujemy wzór:

$$z' = z \cdot e^{i\theta}.$$

Części rzeczywistą i urojoną potrzebne do przedstawienia liczby z' w postaci algebraicznej wyodrębniamy za pomocą funkcji **real()** i **imag()**. W języku **Python** możemy stosować składnię postaci `a, b = 1, 2`, która pozwala w jednej linii kodu jednocześnie przypisać wartość 1 do zmiennej `a` oraz liczbę 2 do zmiennej `b`. Z kolei podczas używania funkcji **show()** należy zastosować cudzysłów z powodu obecności apostrofu w tekście:

```
theta = pi/4
obrot = z * exp(i*theta)
x, y = real(obrot), imag(obrot)
```

```
print('Liczba zespolona po wykonaniu obrotu:')
show( LatexExpr("z'="), x,
 LatexExpr(r'+i\left('), y, LatexExpr(r'\right)') )
```
Out: Liczba zespolona po wykonaniu obrotu:

$$z' = -\frac{1}{2}\sqrt{3}\sqrt{2} - \frac{1}{2}\sqrt{2} + i\left(\frac{1}{2}\sqrt{3}\sqrt{2} - \frac{1}{2}\sqrt{2}\right)$$

d) Na początku kąt $\theta = 75°$ wyrażony w stopniach zapisujemy w radianach:

$$\theta = 75° = 75° \cdot \frac{\pi}{180°} = \frac{5\pi}{12}.$$

Aby wykonać obrót liczby z o kąt θ zgodnie z ruchem wskazówek zegara, stosujemy wzór:

$$z' = z \cdot e^{-i\theta}.$$

Zapisujemy liczbę z' w postaci algebraicznej (3.1), korzystając ze wzoru Eulera (3.6):

$$z' = 2 \cdot e^{i\frac{2\pi}{3}} \cdot e^{-i\frac{5\pi}{12}} = 2 \cdot e^{i\frac{\pi}{4}} = 2 \cdot \left(\cos\frac{\pi}{4} + i\sin\frac{\pi}{4}\right)$$

$$= 2 \cdot \left(\frac{\sqrt{2}}{2} + i \cdot \frac{\sqrt{2}}{2}\right) = \sqrt{2} + i\sqrt{2}.$$

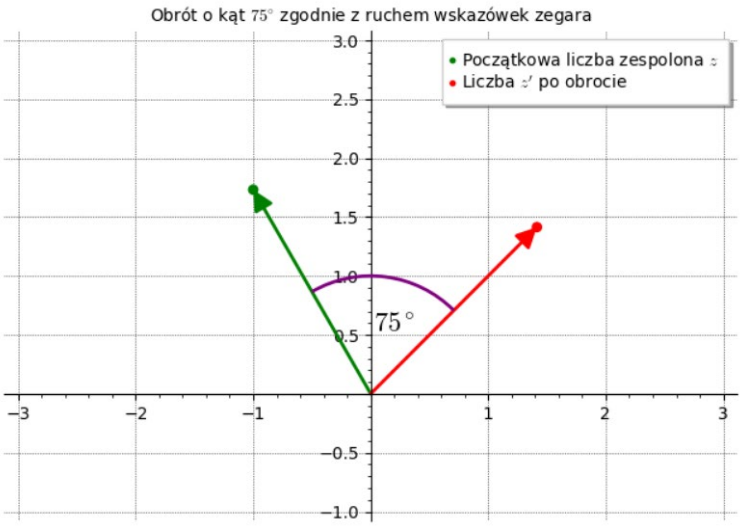
W **Sage** stosujemy funkcję **canonicalize_radical()**, która pozwala uprościć wyrażenia zawierające pierwiastki:

```
theta = 75 * pi/180
obrot = z * exp(-i*theta)

print('Liczba zespolona po wykonaniu obrotu:')
show( LatexExpr("z'="), obrot.canonicalize_radical() )
```
Out: Liczba zespolona po wykonaniu obrotu:

$$z' = (i+1)\sqrt{2}$$

Poniższą wizualizację obrotu na płaszczyźnie zespolonej przedstawiono na rysunku 4.1:

```
rys = plot( vector(z.n()), color='green', gridlines=True,
 title=r'Obrót o kąt $75^\circ$ ' +
 'zgodnie z ruchem wskazówek zegara' )
rys += plot( vector(obrot.n()), color='red' )
rys += point( z, color='green', size=40, zorder=5,
 legend_label='Początkowa liczba zespolona $z$')
rys += point( obrot, color='red', size=40, zorder=5,
 legend_label="Liczba $z'$ po obrocie" )
rys += arc( (0, 0), 1, sector=(arg(obrot), arg(z)),
 thickness=2, color='purple' )
rys += text( r'$75^\circ$', (0.2, 0.6),
 color='black', fontsize=16 )

rys.axes_range(-3, 3, -1, 3)
rys
```



Rysunek 4.1. Obrót liczby zespolonej $-1 + i\sqrt{3}$ o kąt 75° zgodnie z ruchem wskazówek zegara

Zadanie 4.2
Na płaszczyźnie zespolonej narysuj zbiory:
a) $A = \{z \in \mathbb{C}: 2 < |z - 2 + i| \leq 5 \ \wedge \ |z + 3i| \leq |z - 4 - i|\}$;
b) $B = \{z \in \mathbb{C}: 1 \leq |z + 3 + i| < 3 \ \wedge \ \mathrm{Re}(i\overline{z} + 2 - 3i) > 1\}$.

a) Wyrażenie w module dla pierwszego ograniczenia, czyli:

$$2 < |z - 2 + i| \leq 5,$$

przyjmuje wartość 0 dla liczby zespolonej $z_0 = 2 - i$. Tym samym nierówność ta opisuje pierścień o środku w punkcie $(2, -1)$ oraz promieniach równych odpowiednio 2 i 5. Obszar ten zaznaczamy na płaszczyźnie zespolonej za pomocą następujących instrukcji:

```
var('x y')
z = x + y*i
rys1 = region_plot( [2 < abs(z-2+i), abs(z-2+i) < 5], (x, -4, 8), (y, -7, 5),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Pierścień', gridlines=True )
rys1 += region_plot( abs(z-2+i) == 5, (x, -4, 8), (y, -7, 5),
 bordercol='black', borderwidth=3 )
```

Moduły występujące w drugim ograniczeniu, czyli:

$$|z + 3i| \leq |z - 4 - i|,$$

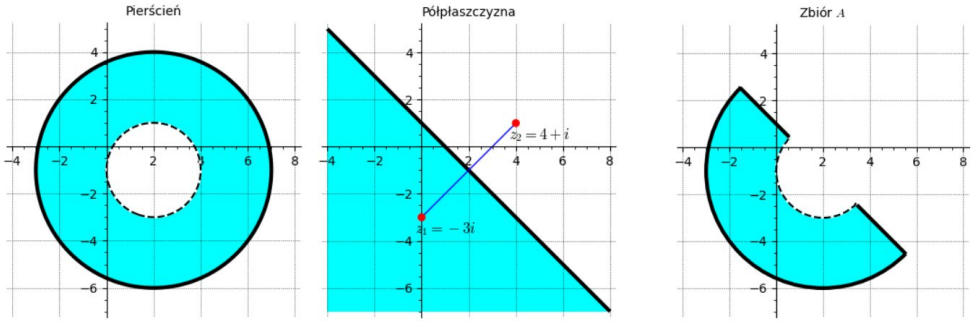
zerowane są przez liczby zespolone – odpowiednio $z_1 = -3i$ oraz $z_2 = 4 + i$. Nierówność ta opisuje półpłaszczyznę leżącą po stronie liczby z_1, która jest ograniczona symetralną odcinka o końcach w punktach $(0, -3)$ i $(4, 1)$. W **Sage** stosujemy polecenia:

```
rys2 = region_plot( [abs(z+3*i) < abs(z-4-i)], (x, -4, 8), (y, -7, 5),
 incol='cyan', bordercol='black', borderwidth=3,
 title='Półpłaszczyzna', gridlines=True )
rys2 += points( [(0, -3), (4, 1)], color='red', size=40, zorder=5 )
rys2 += line( [(0, -3), (4, 1)] )
rys2 += text( '$z_1=-3i$', (1, -3.5), color='black', fontsize=12 )
rys2 += text( '$z_2=4+i$', (5, 0.5), color='black', fontsize=12 )
```

Zbiór A jest częścią wspólną tych dwóch obszarów. Aby go narysować, łączymy odpowiednio wcześniejsze fragmenty kodu:

```
rys3 = region_plot( [2 < abs(z-2+i), abs(z-2+i) < 5,
 abs(z+3*i) < abs(z-4-i)], (x, -4, 8), (y, -7, 5),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $A$', gridlines=True )
rys3 += region_plot( abs(z+3*i) == abs(z-4-i), (x, -1.55, 0.55), (y, -7, 5),
 bordercol='black', borderwidth=3 )
rys3 += region_plot( abs(z+3*i) == abs(z-4-i), (x, 3.45, 5.55), (y, -7, 5),
 bordercol='black', borderwidth=3 )
rys3 += arc( (2, -1), 5, sector=(3*pi/4, 7*pi/4),
 color='black', thickness=3 )

rys1.show(figsize=5)
rys2.show(figsize=5)
rys3.show(figsize=5)
```

Na rysunku 4.2 pokazano kolejne etapy powstawania końcowej ilustracji zbioru A. Najpierw umieszczamy wszystkie ograniczenia w liście będącej pierwszym argumentem funkcji **region_plot()** i zaznaczamy brzeg tego obszaru linią przerywaną. Następnie nadpisujemy brzegi należące do tego zbioru linią ciągłą. Stosujemy funkcję **arc()**, aby zaznaczyć odpowiedni łuk należący do tego obszaru. Wartości liczbowe występujące w poszczególnych funkcjach dobieramy tak jak zwykle – metodą prób i błędów.



Rysunek 4.2. Etapy powstawania zbioru A – pierścień, półpłaszczyzna oraz część wspólna tych obszarów

b) Analogicznie jak w podpunkcie a), pierwsze ograniczenie, czyli:

$$1 \leq |z + 3 + i| < 3,$$

opisuje pierścień, który w tym przypadku jest domknięty wewnętrznie. Wyrażenie w module przyjmuje wartość 0 dla liczby $z_0 = -3 - i$, dlatego jest to pierścień o środku w punkcie $(-3, -1)$ oraz promieniach równych odpowiednio 1 i 3. Lewą stronę nierówności występującej w drugim ograniczeniu, czyli:

$$\mathrm{Re}(i\bar{z} + 2 - 3i) > 1,$$

przekształcamy równoważnie, stosując postać algebraiczną $z = x + yi$ liczby zespolonej z, gdzie $x, y \in \mathbb{R}$. Otrzymujemy:

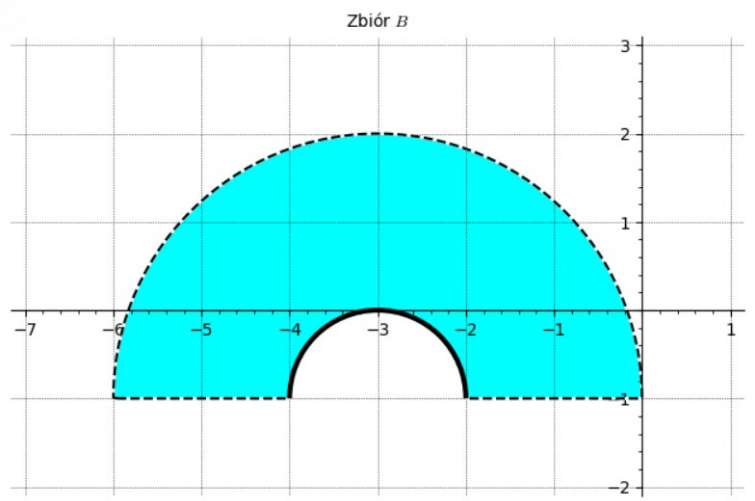
$$\mathrm{Re}[i(x - yi) + 2 - 3i] = \mathrm{Re}[(y + 2) + i(x - 3)] = y + 2.$$

W konsekwencji warunek ten opisuje półpłaszczyznę daną nierównością:

$$y + 2 > 1 \ \rightarrow \ y > -1.$$

Zbiór B, będący częścią wspólną pierścienia i półpłaszczyzny, zaznaczamy na płaszczyźnie zespolonej za pomocą poniższego kodu. Wynik przedstawiono na rysunku 4.3.

```
rys = region_plot( [1 < abs(z+3+i), abs(z+3+i) < 3,
 real(i*conjugate(z) + 2-3*i) > 1], (x, -7, 1), (y, -2, 3),
 incol='cyan', bordercol='black', borderstyle='--',
 title='Zbiór $B$', gridlines=True )
rys += arc( (-3, -1), 1, sector=(0, pi),
 color='black', thickness=3 )
rys
```



Rysunek 4.3. Zbiór B – część wspólna pierścienia o środku w punkcie $(-3,-1)$ i półpłaszczyzny $y > -1$

Zadanie 4.3

Zaznacz na płaszczyźnie zespolonej poniższe zbiory:

a) $A = \left\{ z \in \mathbb{C}\colon \frac{7\pi}{6} < \arg(3iz) < 2\pi \right\}$;

b) $B = \left\{ z \in \mathbb{C}\colon \frac{\pi}{6} \leq \arg(z^3) \leq \frac{5\pi}{6} \right\}$.

a) Wykorzystując wzór (3.7), zgodnie z którym argument iloczynu liczb zespolonych jest sumą ich argumentów (z dokładnością do 2π), otrzymujemy:

$$\arg(3i \cdot z) = \arg(3i) + \arg(z) + 2k\pi = \frac{\pi}{2} + \arg z + 2k\pi, \quad \text{gdzie } k \in \mathbb{Z}.$$

W związku z tym podwójną nierówność opisującą zbiór A możemy zapisać równoważnie jako koniunkcję dwóch nierówności:

$$\frac{7\pi}{6} < \frac{\pi}{2} + \arg z + 2k\pi < 2\pi$$
$$\rightarrow \frac{2\pi}{3} + 2k\pi < \arg z \ \wedge\ \arg z < \frac{3\pi}{2} + 2k\pi \ (k \in \mathbb{Z}).$$

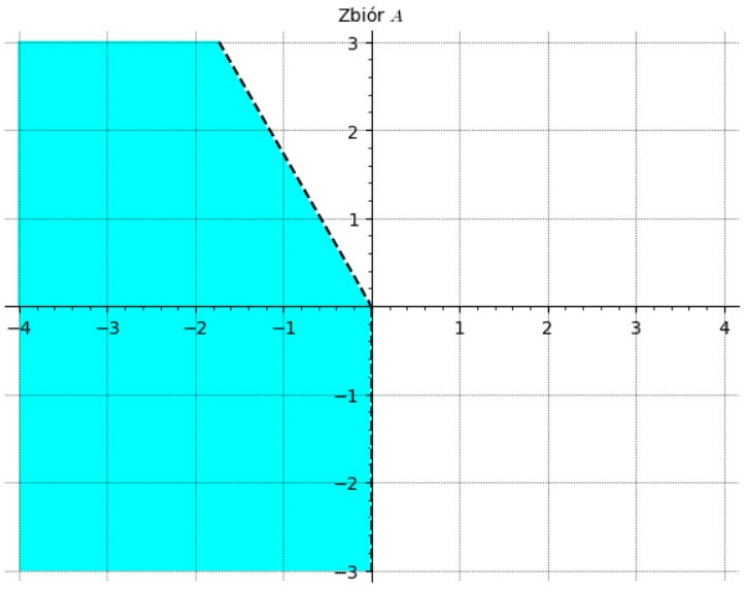
Dopasowujemy wartości parametru $k \in \mathbb{Z}$ tak, aby uzyskać argument główny z zakresu $(-\pi, \pi]$, zgodnego z konwencją stosowaną w **Sage**:

$$k = 0\colon \frac{2\pi}{3} < \operatorname{Arg} z \leq \pi,$$

$$k = -1\colon\ -\pi < \operatorname{Arg} z < -\frac{\pi}{2}.$$

Obszar opisywany przez powyższe nierówności przedstawiamy na rysunku 4.4. W kodzie wykorzystane zostały dwie funkcje **region_plot()**, ponieważ zbiór A rozciąga się na drugą i trzecią ćwiartkę:

```
var('x y')
z = x + y*i
rys = region_plot( 2*pi/3 < arg(z), (x, -4, 4), (y, 0, 3),
 incol='cyan', bordercol='black',
 borderstyle='--', title='Zbiór $A$' )
rys += region_plot( arg(z) < -pi/2, (x, -4, 4), (y, -3, -.01),
 incol='cyan', bordercol='black',
 borderstyle='--', gridlines=True )
rys
```



Rysunek 4.4. Rozwiązanie podpunktu a) – liczby spełniające nierówność $\frac{7\pi}{6} < \arg(3iz) < 2\pi$

b) Ponownie rozpoczynamy od przekształcenia argumentu liczby z^3, czyli iloczynu trzech liczb zespolonych, na sumę ich argumentów (z dokładnością do 2π). Uzyskujemy zależność:

$$\arg(z^3) = \arg(z \cdot z \cdot z) = \arg z + \arg z + \arg z + 2k\pi = 3\arg z + 2k\pi,$$

gdzie $k \in \mathbb{Z}$.

Przekształcamy podwójną nierówność opisującą zbiór B do postaci równoważnej:

$$\frac{\pi}{6} \leq 3\arg z + 2k\pi \leq \frac{5\pi}{6} \rightarrow \frac{\pi}{18} + \frac{2}{3}k\pi \leq \arg z \leq \frac{5\pi}{18} + \frac{2}{3}k\pi \ (k \in \mathbb{Z}).$$

Dobieramy wartości parametru $k \in \mathbb{Z}$, tak aby argument główny mieścił się w przedziale $(-\pi, \pi]$. W tym przypadku otrzymujemy więcej możliwości:

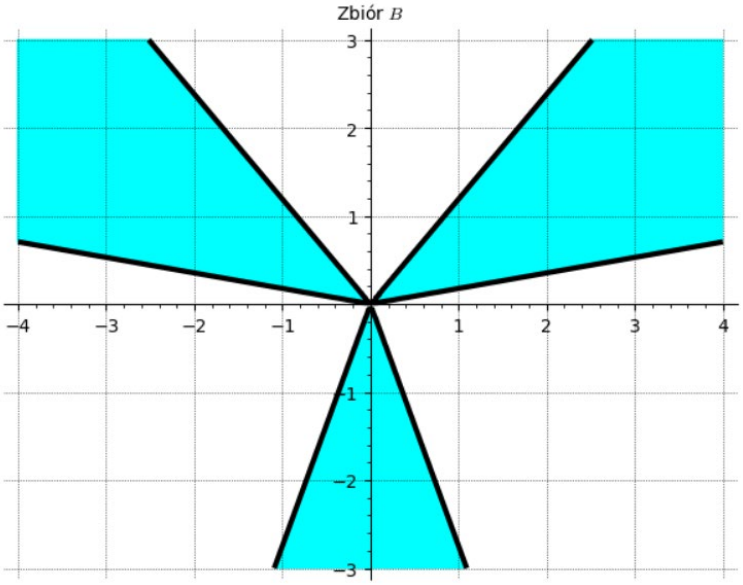
$$k = -1\text{: } -\frac{11\pi}{18} \leq \operatorname{Arg} z \leq -\frac{7\pi}{18},$$

$$k = 0\text{: } \frac{\pi}{18} \leq \operatorname{Arg} z \leq \frac{5\pi}{18},$$

$$k = 1\text{: } \frac{13\pi}{18} \leq \operatorname{Arg} z \leq \frac{17\pi}{18}.$$

Końcowy wynik prezentujemy na rysunku 4.5. W implementacji w **Sage** wystarczy bezpośrednio umieścić początkową nierówność wewnątrz funkcji **region_plot()**:

```
region_plot( [pi/6 < arg(z^3), arg(z^3) < 5*pi/6], (x, -4, 4), (y, -3, 3),
 incol='cyan', bordercol='black', borderwidth=3,
 title='Zbiór $B$', gridlines=True, plot_points=200 )
```



Rysunek 4.5. Rozwiązanie podpunktu b) – obszar dla nierówności $\frac{\pi}{6} \leq \arg(z^3) \leq \frac{5\pi}{6}$

Zadanie 4.4

Znajdź dokładne wartości elementów podanych zbiorów. Następnie zaznacz uzyskane liczby na płaszczyźnie zespolonej:

a) $A = \sqrt{-3+4i}$;

b) $B = \sqrt[3]{2i}$;

c) $C = \{z \in \mathbb{C}: (z+1)^4 = (2+2i)^4\}$.

a) W celu wyznaczenia pierwiastków liczby zespolonej stosujemy wzory (3.9), aby znaleźć jeden z pierwiastków $\omega_0 = x + yi$.

Otrzymujemy:

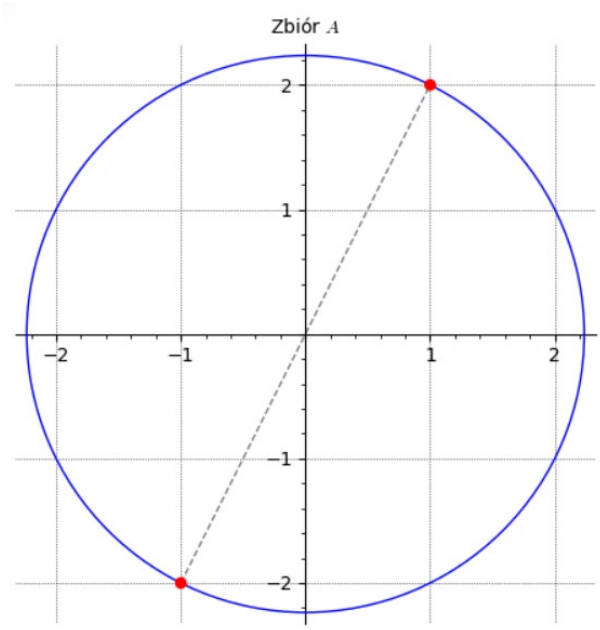
$$x = \sqrt{\frac{-3+\sqrt{(-3)^2+4^2}}{2}} = 1, \quad y = \frac{4}{2 \cdot 1} = 2.$$

Pierwiastek kwadratowy $\sqrt{z}$ składa się z dwóch elementów $\{-\omega_0, \omega_0\}$, czyli:

$$\sqrt{-3+4i} = \{-1-2i, 1+2i\}.$$

W rezultacie otrzymujemy liczby przeciwne, które na płaszczyźnie zespolonej tworzą punkty symetryczne względem początku układu współrzędnych. Ilustrację przedstawiamy na rysunku 4.6.

```
z = -3 + 4*i
x = sqrt( (real(z) + abs(z)) / 2 )
y = imag(z) / (2*x)
w0 = x + y*i
rys = points( [-w0, w0], color='red', size=40, zorder=7,
 title='Zbiór $A$', gridlines=True )
rys += circle( (0, 0), abs(w0), aspect_ratio=1 )
rys += line( [-w0, w0], color='gray', linestyle='--' )
rys
```



Rysunek 4.6. Elementy zbioru $\sqrt{-3+4i}$ zaznaczone kolorem czerwonym

b) Wyznaczamy moduł i argument główny liczby zespolonej $z = 2i$ jako:

$$|z| = 2, \qquad \varphi = \frac{\pi}{2}.$$

Korzystając ze wzoru (3.11), obliczamy jeden z pierwiastków trzeciego stopnia:

$$\omega_0 = \sqrt[3]{|z|} \cdot e^{i\frac{\varphi}{n}} = \sqrt[3]{2} \cdot e^{i\frac{\pi}{6}}.$$

Po zastosowaniu wzoru Eulera (3.6) uzyskujemy postać algebraiczną tego pierwiastka:

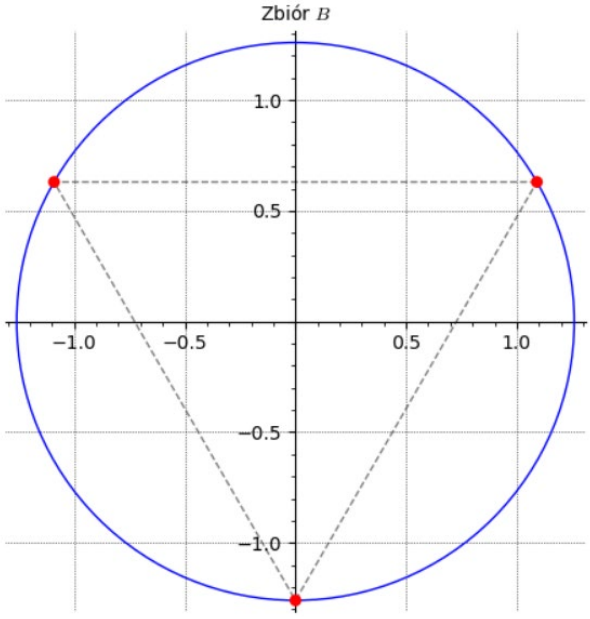
$$\omega_0 = \sqrt[3]{2} \cdot \left(\cos\frac{\pi}{6} + i\sin\frac{\pi}{6}\right) = \sqrt[3]{2} \cdot \left(\frac{\sqrt{3}}{2} + \frac{1}{2}i\right).$$

Pozostałe pierwiastki otrzymujemy, mnożąc przez czynnik $\zeta = e^{i\frac{2\pi}{3}}$. Są to liczby:

$$\omega_1 = \sqrt[3]{2} \cdot e^{i\frac{5\pi}{6}} = \sqrt[3]{2} \cdot \left(\cos\frac{5\pi}{6} + i\sin\frac{5\pi}{6}\right) = \sqrt[3]{2} \cdot \left(-\frac{\sqrt{3}}{2} + \frac{1}{2}i\right)$$

$$\omega_2 = \sqrt[3]{2} \cdot e^{i\frac{9\pi}{6}} = \sqrt[3]{2} \cdot \left(\cos\frac{3\pi}{2} + i\sin\frac{3\pi}{2}\right) = \sqrt[3]{2} \cdot (0 - i) = -\sqrt[3]{2}i.$$

Interpretacją geometryczną elementów pierwiastka trzeciego stopnia są wierzchołki trójkąta równobocznego. Przedstawiamy je na rysunku 4.7, wykorzystując kod z przykładu 3.23.



Rysunek 4.7. Elementy zbioru $\sqrt[3]{2i}$ tworzą wierzchołki trójkąta równobocznego

```
z, n = 2*i, 3
modul = abs(z)
phi = arg(z)
w0 = modul^(1/n) * exp(i*phi/n)
pierwiastki = [w0 * exp(i*2*k*pi/n) for k in range(n)]

rys = points( pierwiastki, color='red', size=40, zorder=7,
 title='Zbiór $B$', gridlines=True )
rys += circle( (0, 0), abs(w0), aspect_ratio=1 )
rys += polygon( pierwiastki, color='gray',
 linestyle='--', fill=False )
rys
```

c) Aby wyznaczyć elementy zbioru

$$C = \{z \in \mathbb{C}: (z+1)^4 = (2+2i)^4\},$$

wprowadzamy podstawienie $\omega = z + 1$. Najpierw znajdujemy liczby zespolone $\omega \in \mathbb{C}$ spełniające poniższe równanie, które zapisujemy równoważnie przy użyciu pierwiastka:

$$\omega^4 = (2+2i)^4 \;\rightarrow\; \omega = \sqrt[4]{(2+2i)^4}.$$

Jeden z pierwiastków odgadujemy jako $\omega_0 = 2 + 2i$, a pozostałe obliczamy poprzez mnożenie kolejnych rozwiązań przez czynnik $\zeta = e^{i\frac{2\pi}{4}}$, który na podstawie obliczeń zawartych w przykładzie 3.22 jest równy $\zeta = i$. Stąd pozostałe pierwiastki to:

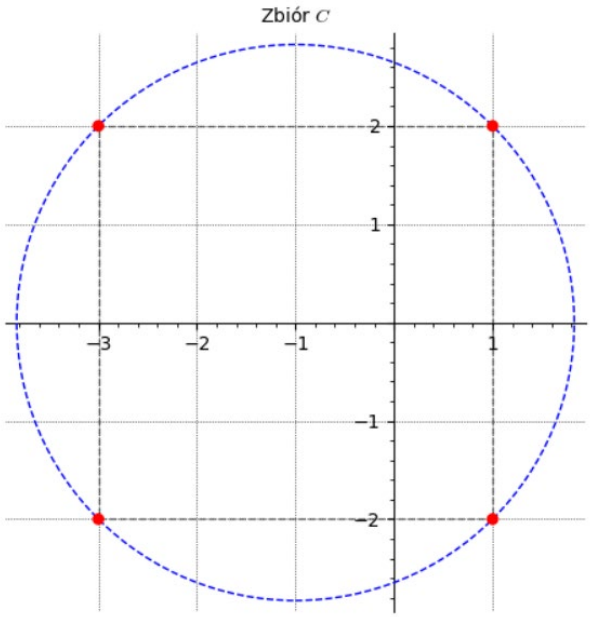
$$\omega_1 = \omega_0 \cdot i = -2 + 2i, \quad \omega_2 = \omega_1 \cdot i = -2 - 2i, \quad \omega_3 = \omega_2 \cdot i = 2 - 2i.$$

Wracamy do podstawienia $z = \omega - 1$, aby wyznaczyć elementy zbioru C. Uzyskane liczby na płaszczyźnie zespolonej są wierzchołkami kwadratu:

$$C = \{1 + 2i, -3 + 2i, -3 - 2i, 1 - 2i\}.$$

W **Sage** zadanie tym razem rozwiązujemy za pomocą funkcji **solve()**:

```
var('z')
rozw = solve( (z+1)^4 == (2+2*i)^4, z )
pierwiastki = [x.rhs() for x in rozw]

rys = points( pierwiastki, color='red', size=40, zorder=7,
 title='Zbiór $C$', gridlines=True )
rys += circle( (-1, 0), 2*sqrt(2), linestyle='--' )
rys += polygon( sorted(pierwiastki, key=lambda x: arg(x)),
 color='gray', linestyle='--', fill=False )
rys
```



Rysunek 4.8. Liczby zespolone $z \in \mathbb{C}$ spełniające równanie $(z+1)^4 = (2+2i)^4$

Wynik został przedstawiony na rysunku 4.8. Funkcja **solve()** domyślnie nie zwraca pierwiastków w kolejności odpowiadającej ich położeniu wzdłuż boków figury, więc bezpośrednie użycie polecenia polygon(pierwiastki) dałoby błędny rezultat. W związku z tym, aby poprawnie narysować wielokąt, jako pierwszy argument funkcji **polygon()** przekazujemy pierwiastki posortowane według argumentów

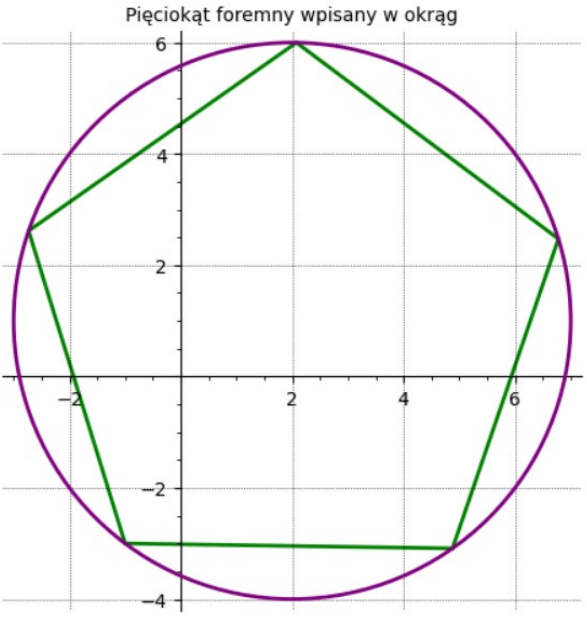
głównych za pomocą funkcji **sorted()**. Wykorzystujemy w tym celu parametr key i funkcję anonimową **lambda** dostępną w języku **Python**.

Zadanie 4.5

Na płaszczyźnie narysuj poniższe figury:
a) pięciokąt foremny wpisany w okrąg o środku w punkcie $S = (2, 1)$;
b) sześciokąt foremny, którego jeden z wierzchołków znajduje się w punkcie $A = (0, 0)$.

a) Wykorzystamy podejście zaprezentowane w przykładzie 3.25. Najpierw przesuwamy pięciokąt tak, aby środek okręgu opisanego znalazł się w punkcie $S' = (0, 0)$. Następnie wybieramy dowolny wierzchołek początkowy, na przykład $A = (-1, -3)$. Wówczas przesunięty punkt ma współrzędne $A' = (-3, -4)$. Kolejne wierzchołki wyznaczamy jako elementy pewnego pierwiastka piątego stopnia, gdzie jednym z elementów tego zbioru jest liczba $\omega_0 = -3 - 4i$. Na końcu wykonujemy transformację odwrotną, przesuwając wszystkie punkty o wektor $\overrightarrow{S'S} = [2, 1]$ (w kodzie dodajemy do każdego pierwiastka liczbę $2 + i$). Wynik działania poniższego kodu przedstawiony został na rysunku 4.9.

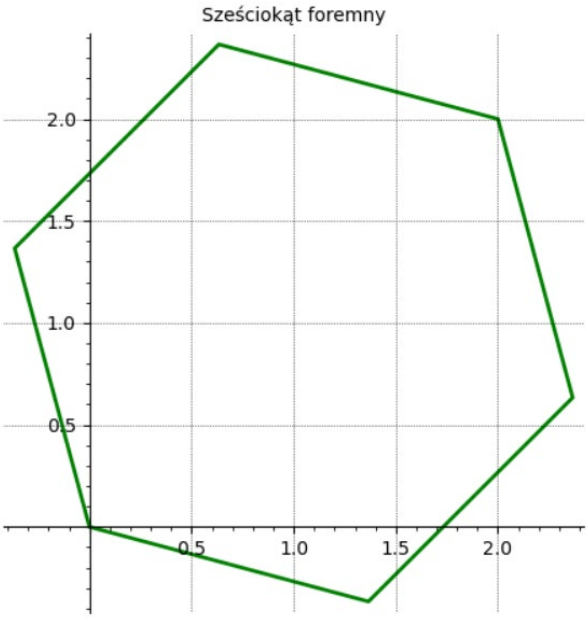
```
A = -1 - 3*i
S = 2 + i
w0 = A - S
n = 5
pierwiastki = [w0 * exp(i*2*k*pi/n) + S for k in range(n)]
rys = polygon( pierwiastki, color='green', thickness=2, aspect_ratio=1,
 fill=False, gridlines=True, title='Pięciokąt foremny wpisany w okrąg' )
rys += circle( (2, 1), 5, color='purple', thickness=2 )
rys
```



Rysunek 4.9. Pięciokąt foremny wpisany w okrąg o środku w punkcie $(2, 1)$

b) Postępujemy analogicznie jak w podpunkcie a), tylko tym razem możemy dowolnie wybrać środek symetrii S sześciokąta. Wybieramy punkt $S = (1, 1)$, co w kodzie zapisujemy jako liczbę zespoloną $1 + i$. Uzyskany rezultat zaprezentowano na rysunku 4.10.

```
A = 0
S = 1 + i
w0 = A - S
n = 6
pierwiastki = [w0 * exp(i*2*k*pi/n) + S for k in range(n)]
polygon( pierwiastki, color='green', thickness=2, aspect_ratio=1,
 fill=False, gridlines=True, title='Sześciokąt foremny' )
```



Rysunek 4.10. Sześciokąt foremny o wierzchołku w punkcie $(0, 0)$

Zadanie 4.6

Na płaszczyźnie narysuj siedem sześciokątów foremnych w taki sposób, aby pierwszy sześciokąt miał wierzchołek w punkcie $(4, 5)$ i środek symetrii w początku układu współrzędnych, natomiast każdy kolejny był wpisany w poprzedni – wierzchołki kolejnego sześciokąta powinny leżeć w środkach boków poprzedniego.

Stosujemy podejście opisane w przykładzie 3.26. Dla sześciokąta foremnego stosunek długości promienia okręgu wpisanego r do promienia okręgu opisanego R wynosi:

$$\sigma = \frac{r}{R} = \frac{\frac{a\sqrt{3}}{2}}{a} = \frac{\sqrt{3}}{2},$$

gdzie a jest długością boku sześciokąta, natomiast miara kąta θ, o który obracamy jego kolejne wierzchołki, jest równa:

$$\theta = \frac{\pi}{n} = \frac{\pi}{6},$$

przy czym $n = 6$ jest liczbą boków sześciokąta. Wynik został przedstawiony na rysunku 4.11.

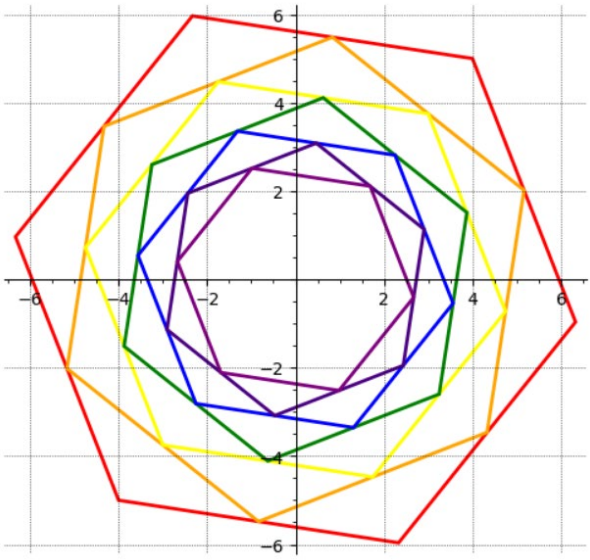
```
w0 = 4 + 5*i
n = 6
s = sqrt(3)/2
theta = pi/6

rys = Graphics()
col = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'purple']
for x in range(7):
    pierwiastki = [w0 * exp(i*2*k*pi/n) *
    exp(i*theta*x) * s^x for k in range(n)]
    rys += polygon( pierwiastki, color=col[x],
    thickness=2, aspect_ratio=1,
    fill=False, gridlines=True )
rys
```

Zadanie 4.7

Rozwiąż równania:

a) $z^2 + (2 + i)z - 1 + 7i = 0$;

b) $z + \overline{5z} + 4i(z - \overline{z}) - 1 + 3i = 0$.



Rysunek 4.11. Ciąg sześciokątów foremnych – wierzchołki kolejnej figury są środkami boków figury poprzedniej

a) Obliczamy wyróżnik trójmianu kwadratowego:

$$\Delta = (2+i)^2 - 4 \cdot 1 \cdot (-1+7i) = 4 + 4i - 1 + 4 - 28i = 7 - 24i.$$

Stosujemy wzory (3.9), aby znaleźć jeden z pierwiastków $\delta = x + yi$ liczby Δ:

$$x = \sqrt{\frac{7 + \sqrt{7^2 + (-24)^2}}{2}} = 4, \qquad y = \frac{-24}{2 \cdot 4} = -3$$

$$\delta = 4 - 3i.$$

Następnie korzystamy ze wzorów (3.12) na pierwiastki równania kwadratowego:

$$z_1 = \frac{-(2+i) - (4-3i)}{2 \cdot 1} = -3 + i, \qquad z_2 = \frac{-(2+i) + (4-3i)}{2 \cdot 1} = 1 - 2i.$$

Wykorzystujemy powyższe wzory, aby rozwiązać równanie w **Sage**. Współczynniki trójmianu kwadratowego odczytujemy za pomocą listy składanej oraz funkcji **coefficients()**:

```
W(z) = z^2 + (2 + i)*z - 1 + 7*i
c, b, a = [w[0] for w in W(z).coefficients()]
delta = b^2 - 4*a*c

x = sqrt( (real(delta) + abs(delta)) / 2 )
y = imag(delta) / (2*x)
d = x + y*i
z1 = (-b-d)/(2*a)
z2 = (-b+d)/(2*a)

print('Rozwiązania równania kwadratowego:')
show( LatexExpr('z_1='), z1 )
show( LatexExpr('z_2='), z2 )
```

Out: Rozwiązania równania kwadratowego:

$z_1 = i - 3$

$z_2 = -2i + 1$

b) Ze względu na to, że w równaniu pojawia się symbol sprzężenia, przedstawiamy liczbę $z \in \mathbb{C}$ w postaci algebraicznej $z = x + yi$, gdzie $x, y \in \mathbb{R}$. Otrzymujemy:

$$(x + yi) + (5x - 5yi) + 4i \cdot [(x + yi) - (x - yi)] - 1 + 3i = 0.$$

Po redukcji wyrazów podobnych uzyskujemy równość:

$$6x - 4yi + 4i \cdot 2yi = 1 - 3i \;\rightarrow\; 6x - 8y - 4yi = 1 - 3i.$$

Dwie liczby zespolone są równe, gdy równe są ich części rzeczywiste i urojone. Stąd uzyskujemy układ równań:

$$\begin{cases} 6x - 8y = 1 \\ -4y = -3. \end{cases}$$

Rozwiązując go, otrzymujemy:

$$\begin{cases} x = \frac{7}{6} \\ y = \frac{3}{4}. \end{cases}$$

Ostatecznie rozwiązaniem równania jest liczba zespolona:

$$z = \frac{7}{6} + \frac{3}{4}i.$$

W **Sage** możemy zastosować kod wykorzystywany w przykładzie 3.29. Tutaj dodatkowo odczytujemy część rzeczywistą i urojoną rozwiązania:

```
var('x y')
z = x + y*i
rownanie = z + conjugate(5*z) + 4*i*(z - conjugate(z)) - 1+3*i

rozw = solve( [rownanie.real(), rownanie.imag()], [x, y] )
x, y = [x.rhs() for x in rozw[0]]
print('Rozwiązaniem jest liczba zespolona:', x+y*i)
Out: Rozwiązaniem jest liczba zespolona: 3/4*I + 7/6
```

Zadanie 4.8

Wyznacz moduł i argument główny liczby zespolonej $z \in \mathbb{C}$ spełniającej równanie:

$$8 \cdot \left(\cos\frac{11\pi}{6} + i\sin\frac{11\pi}{6}\right) \cdot (1-i)^{10} \cdot z = 64 \cdot e^{i\frac{5\pi}{3}}.$$

Wprowadzamy oznaczenia:

$$z_1 = 8 \cdot \left(\cos\frac{11\pi}{6} + i\sin\frac{11\pi}{6}\right), \qquad z_2 = (1-i)^{10}, \qquad z_3 = 64 \cdot e^{i\frac{5\pi}{3}}.$$

Wyznaczamy moduł i argument każdej z tych liczb zespolonych. Liczba z_1 zapisana jest w postaci trygonometrycznej (3.5), więc mamy:

$$|z_1| = 8, \qquad \varphi_1 = \frac{11\pi}{6}.$$

Dla liczby z_2 najpierw zapisujemy wartość $\zeta = 1 - i$ w postaci wykładniczej. Moduł wyznaczamy ze wzoru (3.3):

$$|\zeta| = \sqrt{1^2 + (-1)^2} = \sqrt{2}.$$

Ponieważ promień wodzący liczby ζ znajduje się w czwartej ćwiartce i tworzy kąt 45° z dodatnią półosią rzeczywistą, jej argument wynosi:

$$\theta = -\frac{\pi}{4}.$$

Zapisujemy liczbę z_2 w postaci wykładniczej (3.4), aby wykonać potęgowanie. Wykorzystujemy wzór (3.8) oraz fakt, że w postaci wykładniczej możemy dodawać do argumentu dowolną wielokrotność liczby 2π:

$$z_2 = \left(\sqrt{2} \cdot e^{-i\frac{\pi}{4}}\right)^{10} = \left(\sqrt{2}\right)^{10} \cdot e^{-i\frac{10\pi}{4}} = 2^5 \cdot e^{-i\frac{\pi}{2}}.$$

Teraz możemy zapisać moduł i argument tej liczby jako:

$$|z_2| = 32, \quad \varphi_2 = -\frac{\pi}{2}.$$

Liczba z_3 już jest zapisana w postaci wykładniczej, zatem wystarczy odczytać interesujące nas wielkości, a mianowicie:

$$|z_3| = 64, \quad \varphi_3 = \frac{5\pi}{3}.$$

Szukana liczba zespolona $z \in \mathbb{C}$ dla wprowadzonych oznaczeń wyrażona jest wzorem:

$$z = \frac{z_3}{z_1 \cdot z_2}.$$

Moduły iloczynu i ilorazu liczb zespolonych odpowiednio mnożymy i dzielimy. Stąd moduł liczby z wynosi:

$$|z| = \frac{|z_3|}{|z_1| \cdot |z_2|} = \frac{64}{8 \cdot 32} = \frac{1}{4}.$$

Argumenty iloczynu i ilorazu liczb zespolonych odpowiednio dodajemy i odejmujemy, wobec czego argument główny liczby z to:

$$\varphi = \varphi_3 - \varphi_1 - \varphi_2 = \frac{5\pi}{3} - \frac{11\pi}{6} - \left(-\frac{\pi}{2}\right) = \frac{\pi}{3}.$$

Poniżej przedstawiamy przykładowe rozwiązanie tego zadania w **Sage**:

```
z1 = 8 * ( cos(11*pi/6) + i*sin(11*pi/6) )
z2 = (1-i)^10
z3 = 64 * exp(i*5*pi/3)

modul_1 = abs(z1).simplify()
modul_2 = abs(z2)
modul_3 = abs(z3).simplify()

phi_1 = arg(z1).simplify()
phi_2 = arg(z2)
phi_3 = arg(z3).simplify()

modul = modul_3 / modul_1 / modul_2
phi = phi_3 - phi_1 - phi_2

print('Moduł liczby zespolonej to:', modul)
print('Argument liczby zespolonej to:', phi)
Out: Moduł liczby zespolonej to: 1/4
 Argument liczby zespolonej to: 1/3*pi
```

Zadanie 4.9

Wiedząc, że liczba $z_0 = 1 + i$ jest pierwiastkiem wielomianu

$$W(z) = z^3 + (3 - i)z^2 + (4 - 4i)z - 8 - 8i,$$

rozłóż ten wielomian na iloczyn dwumianów.

Wielomian W dzielimy przez dwumian $\big(z - (1 + i)\big)$ za pomocą schematu Hornera:

	1	$3 - i$	$4 - 4i$	$-8 - 8i$
$z_0 = 1 + i$	$\downarrow$	$1 + i$	$4 + 4i$	$8 + 8i$
	1	4	8	0

Uzyskujemy następujący rozkład na czynniki:

$$W(z) = (z - 1 - i) \cdot (z^2 + 4z + 8).$$

Następnie obliczamy wyróżnik powstałego trójmianu kwadratowego:

$$\Delta = 4^2 - 4 \cdot 1 \cdot 8 = -16 = (4i)^2.$$

Korzystając ze wzorów (3.12) na pierwiastki trójmianu, otrzymujemy:

$$z_1 = \frac{-4 - 4i}{2 \cdot 1} = -2 - 2i, \quad z_2 = \frac{-4 + 4i}{2 \cdot 1} = -2 + 2i.$$

Ostateczny rozkład na czynniki liniowe ma postać:

$$W(z) = (z - 1 - i)(z + 2 + 2i)(z + 2 - 2i).$$

W **Sage** prezentujemy alternatywne podejście, deklarując wielomian nad ciałem liczb zespolonych. W tym celu używamy instrukcji `R.<z> = CC[]`, gdzie obiekt `CC` jest tożsamy z ciałem `ComplexField(prec=53)`. Dla zwiększenia czytelności wyniku w pierwszej linii kodu nadpisujemy go obiektem o niższej precyzji:

```
CC = ComplexField(prec=8)
R.<z> = CC[]
W = z^3 + (3-i)*z^2 + (4-4*i)*z - 8-8*i
factor(W)
Out: (z - 1.0 - I) * (z + 2.0 - 2.0*I) * (z + 2.0 + 2.0*I)
```

4.2. Zadania do samodzielnego rozwiązania

Zadanie 4.10

Dane są liczby zespolone:

$$z = -\sqrt{2} + i\sqrt{2} \quad \text{oraz} \quad \omega = 6 + 8i.$$

Oblicz i zapisz w najprostszej postaci liczby:

a) $\frac{\mathrm{Re}(\omega^2)}{|z|}$; b) $\left|\frac{\overline{z}}{2\omega}\right|$; c) $\mathrm{Arg}\left(\frac{\mathrm{Im}\,\omega}{z}\right)$.

Zakładamy, że argument główny należy do przedziału $[0, 2\pi)$.

Zadanie 4.11

Wyznacz sprzężenie liczby z oraz zaznacz liczby z i $\overline{z}$ na płaszczyźnie zespolonej:

a) $z = 2i$; b) $z = -3 - 2i$.

Zadanie 4.12

Na płaszczyźnie zespolonej zaznacz liczbę z oraz liczbę z' uzyskaną w wyniku obrotu liczby z o kąt θ przeciwnie do ruchu wskazówek zegara:

a) $z = \sqrt{2} + i\sqrt{2}, \theta = 75°$; b) $z = \sqrt{3} - i, \theta = 120°$.

Zadanie 4.13

Na płaszczyźnie zespolonej zaznacz liczbę z oraz liczbę z' uzyskaną w wyniku obrotu liczby z o kąt θ zgodnie z ruchem wskazówek zegara:

a) $z = 1 + 2i, \theta = 105°$; b) $z = -1 + i\sqrt{3}, \theta = 150°$.

Zadanie 4.14

Podaj dokładną wartość modułu i argumentu głównego należącego do przedziału $[0, 2\pi)$ dla liczby zespolonej z:

a) $z = \frac{7+i}{3+4i}$; b) $z = \frac{3i-7}{2-5i}$;

c) $z = \frac{\mathrm{Im}(5+4i)}{i\sqrt{2}-\sqrt{2}}$; d) $z = \frac{\mathrm{Re}(i-4)}{i+\sqrt{3}}$;

e) $z = \frac{\mathrm{Re}(-1+3i)+\mathrm{Im}(-2+5i)}{\sqrt{2}+i\sqrt{2}}$; f) $z = \frac{\mathrm{Re}(5-4i)+\mathrm{Im}(-2+3i)}{\sqrt{3}+i}$.

Zadanie 4.15

Na płaszczyźnie zespolonej narysuj zbiór:

a) $A = \{z \in \mathbb{C}: \mathrm{Im}[(1-i)\overline{z} + 4 + 3i] > 2\}$;

b) $B = \{z \in \mathbb{C}: \mathrm{Re}[(1+2i)\overline{z} + 2 - 3i] < 1\}$;

c) $C = \{z \in \mathbb{C}: \mathrm{Re}(z \cdot \overline{z}) > 4\}$;

d) $D = \{z \in \mathbb{C}: |z - 3 + i| \leq |z + 3 - 4i|\}$;

e) $E = \left\{z \in \mathbb{C}: 1 < |z - 1| \leq \left|1 - i\sqrt{3}\right|\right\}$;

f) $F = \{z \in \mathbb{C}: 2 \leq |z + 1 + i| < |3 - i|\}$;

g) $G = \{z \in \mathbb{C}: |z - 1 + i| < |z + 1 - i| \leq |z + 3 - i|\}$;

h) $H = \{z \in \mathbb{C}: \operatorname{Re}(z\overline{z}) \leq 9 \wedge \operatorname{Im}(zi) > 0\}$;

i) $I = \{z \in \mathbb{C}: |z + 3 + i| \leq 3 \wedge \operatorname{Re}(i\overline{z} + 2 - 3i) > 1\}$;

j) $J = \{z \in \mathbb{C}: \operatorname{Re} z > 1 \wedge \operatorname{Im} z > 1 \wedge |z - 1 - i| \leq 3\}$.

Zadanie 4.16

Znajdź dokładne wartości elementów podanych zbiorów. Zaznacz znalezione pierwiastki na płaszczyźnie zespolonej:

a) $A = \sqrt{15 + 8i}$; b) $B = \sqrt[3]{-i}$; c) $C = \sqrt[4]{-3 + 4i}$;

d) $D = \sqrt[5]{(3 - 2i)^5}$; e) $E = \sqrt[6]{1 - i}$; f) $F = \sqrt[8]{1}$.

Zadanie 4.17

Narysuj n-kąt foremny, taki że jednym z jego wierzchołków jest punkt A, a środkiem okręgu opisanego na tym wielokącie jest punkt S:

a) $n = 3, A = (0, 0), S = (3, 3)$;

b) $n = 8, A = (2, 3), S = (2, 0)$;

c) $n = 16, A = (1, 1), S = (-1, -1)$.

Zadanie 4.18

W ciele liczb zespolonych rozwiąż równanie:

a) $\frac{3z}{1-3i} = (1 + 3i)^3$; b) $i(z + 3\overline{z}) = 1 - 2i$;

c) $3z - 2\overline{z} = (1 + 2i)^2$; d) $2z - 3\overline{z} + i(z - i) = 4 - i$.

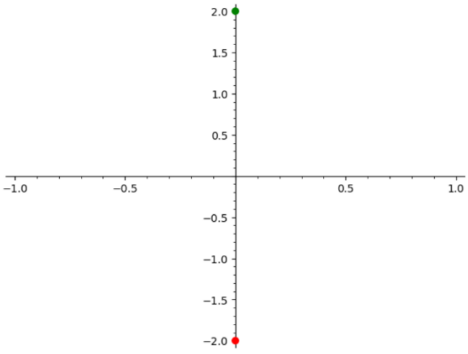
4.3. Odpowiedzi do zadań

Zadanie 4.10

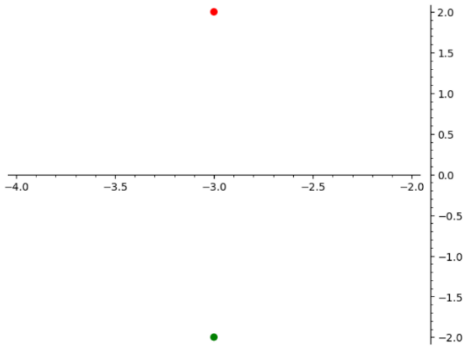
a) –14; b) $\frac{1}{10}$; c) $\frac{5\pi}{4}$.

Zadanie 4.11

a)

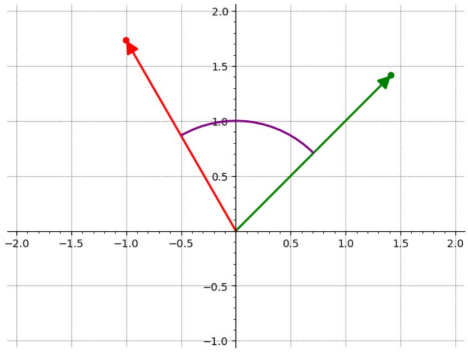


b)

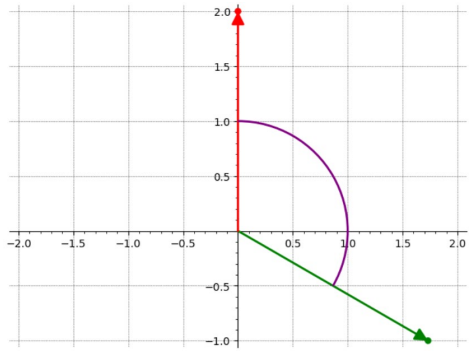


Zadanie 4.12

a)

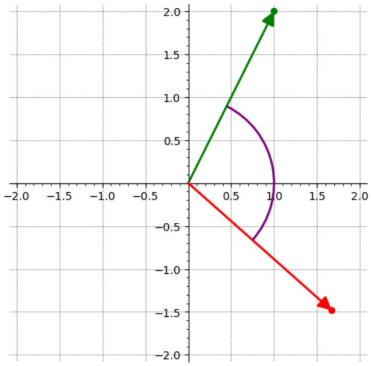


b)

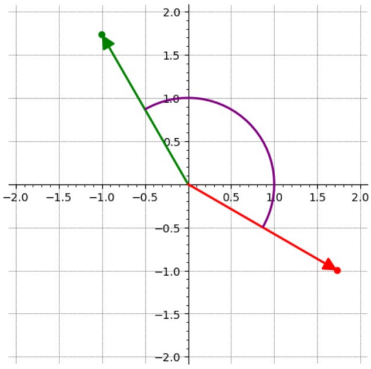


Zadanie 4.13

a)



b)



Zadanie 4.14

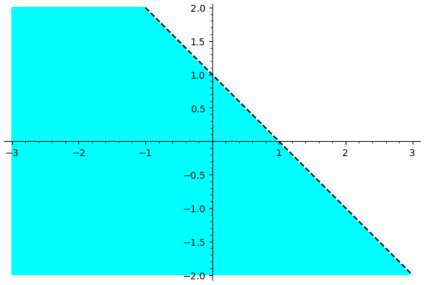
a) $|z| = \sqrt{2}, \operatorname{Arg} z = \frac{7\pi}{4}$; b) $|z| = \sqrt{2}, \operatorname{Arg} z = \frac{5\pi}{4}$;

c) $|z| = 2, \operatorname{Arg} z = \frac{5\pi}{4}$; d) $|z| = 2, \operatorname{Arg} z = \frac{5\pi}{6}$;

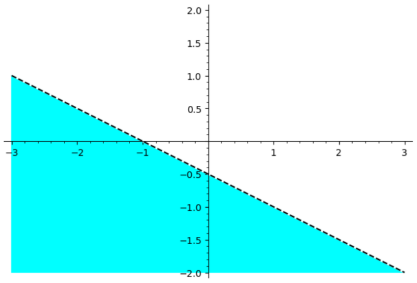
e) $|z| = 2, \operatorname{Arg} z = \frac{7\pi}{4}$; f) $|z| = 4, \operatorname{Arg} z = \frac{11\pi}{6}$.

Zadanie 4.15

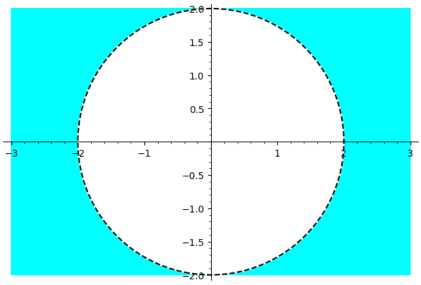
a)



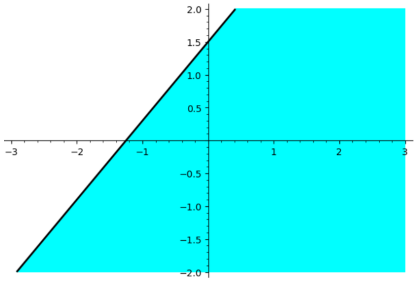
b)



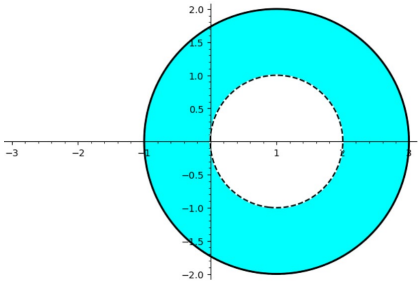
c)



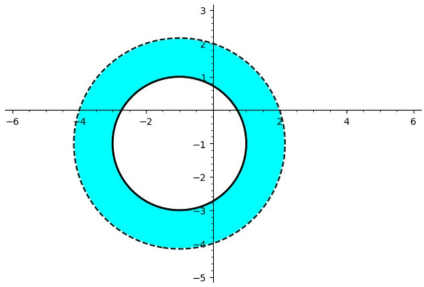
d)



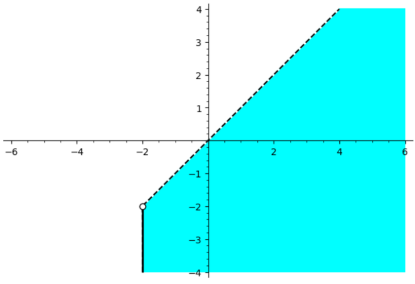
e)



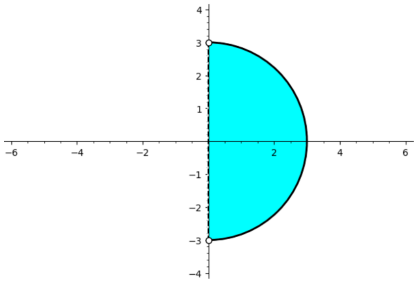
f)



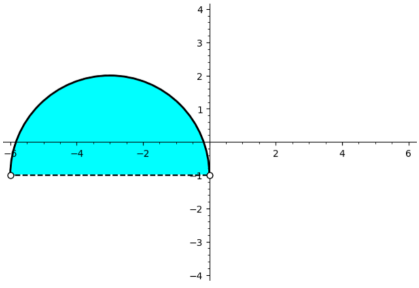
g)



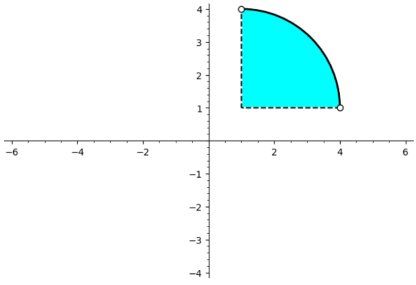
h)



i)

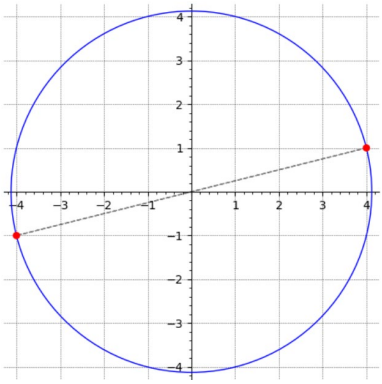


j)

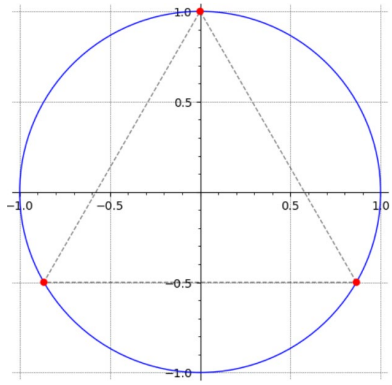


Zadanie 4.16

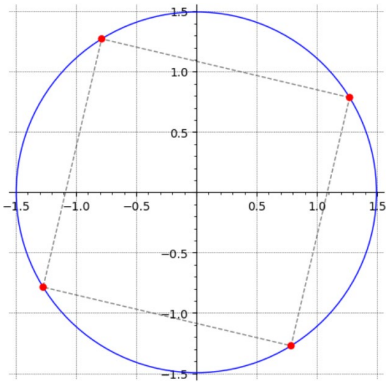
a)



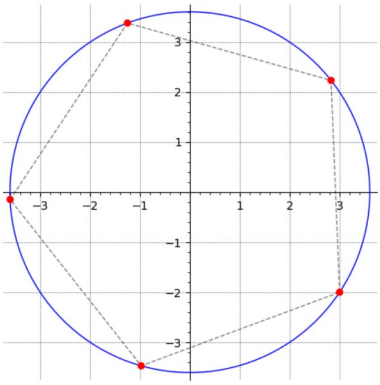
b)



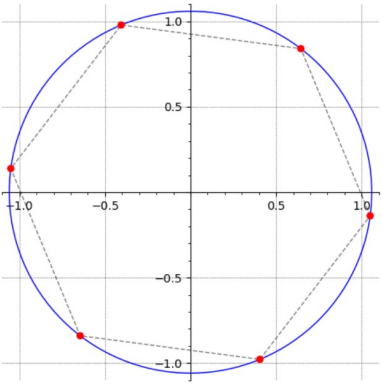
c)



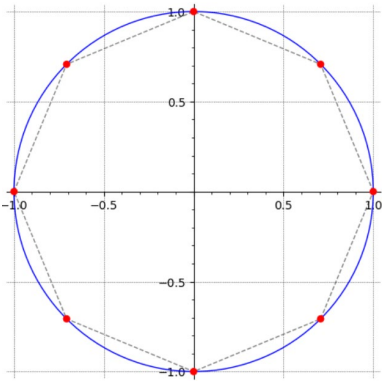
d)



e)

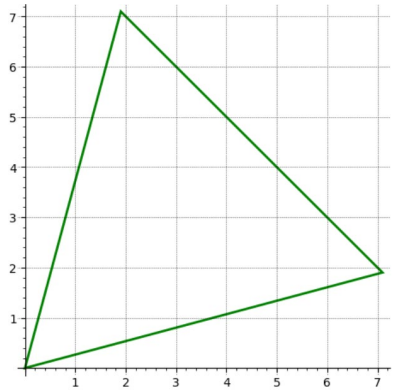


f)

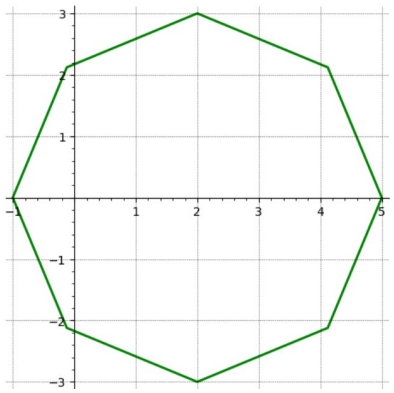


Zadanie 4.17

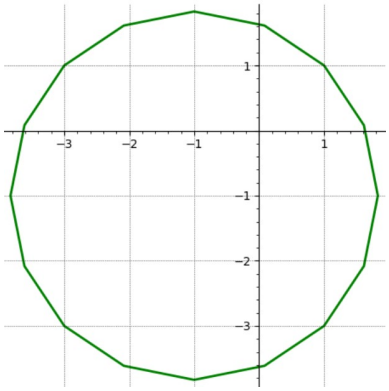
a)



b)



c)



Zadanie 4.18

a) $z = -\frac{80}{3} + 20i$; b) $z = -\frac{1}{2} + \frac{1}{2}i$;

c) $z = -3 + \frac{4}{5}i$; d) $z = -\frac{7}{2} + \frac{1}{2}i$.

Wszystkiego dobrego! 😉

Dodatek A.
Spis funkcji występujących w książce

Nazwa funkcji	Opis
abs()	moduł liczby zespolonej
acos()	arcus cosinus
acot()	arcus cotangens
arc()	rysuje łuk okręgu
arg()	argument liczby zespolonej
asin()	arcus sinus
assume()	ustawia założenie o zmiennej
assumptions()	wyświetla aktualne założenia
atan()	arcus tangens
axes_range()	ustawia zakres osi na wykresie
binomial()	symbol Newtona
canonicalize_radical()	upraszcza wyrażenia z pierwiastkami
CDF()	polecenie do obliczeń na liczbach zespolonych
ceil()	zaokrągla w górę do najbliższej liczby całkowitej
circle()	rysuje okrąg
coefficient()	zwraca wybrany współczynnik wielomianu
coefficients()	zwraca listę wszystkich współczynników
ComplexField()	ciało liczb zespolonych o zadanej precyzji
conjugate()	sprzężenie liczby zespolonej
cos()	cosinus
cot()	cotangens
divisors()	lista wszystkich dzielników liczby
exp()	eksponenta (funkcja wykładnicza o podstawie e)
expand()	zapisuje wyrażenie w postaci ogólnej
factor()	zapisuje wyrażenie w postaci iloczynowej
factorial()	silnia
find_root()	znajduje przybliżone rozwiązanie równania
floor()	zaokrągla w dół do najbliższej liczby całkowitej
forget()	usuwa wcześniejsze założenia
gcd()	największy wspólny dzielnik (NWD)

Graphics()	tworzy pusty obiekt graficzny 2D
imag()	część urojona liczby zespolonej
LatexExpr()	konwertuje tekst na zapis w języku **LaTeX**
lcm()	najmniejsza wspólna wielokrotność (NWW)
lhs()	lewa strona równości
line()	rysuje odcinek
ln()	logarytm naturalny
log()	logarytm (domyślnie naturalny, ale można zmienić podstawę)
n()	przybliżenie numeryczne
nth_root()	pierwiastek n-tego stopnia
plot()	rysuje wykres funkcji
point()	rysuje pojedynczy punkt
points()	rysuje wiele punktów
polygon()	rysuje wielokąt
print()	wypisuje na ekran
range()	zakres liczb całkowitych
real()	część rzeczywista liczby zespolonej
region_plot()	rysuje obszar na płaszczyźnie
reset()	przywraca środowisko do początkowego stanu
rhs()	prawa strona równości
round()	zaokrąglenie liczby
save()	zapisuje obiekt do pliku
show()	wyświetla obiekt
simplify()	upraszcza wyrażenie ogólnie
simplify_full()	upraszcza wyrażenie maksymalnie
simplify_rational()	upraszcza ułamki algebraiczne
simplify_trig()	upraszcza wyrażenia trygonometryczne
sin()	sinus
solve()	rozwiązuje równanie, nierówność lub układ równań
sorted()	sortuje elementy listy
sqrt()	pierwiastek kwadratowy
tan()	tangens
text()	dodaje napis do wykresu
var()	deklaruje zmienne symboliczne
vector()	tworzy wektor

Literatura

1. Bard G.V., *Sage for undergraduates*, American Mathematical Society, Providence 2015.
2. Jurlewicz T., Skoczylas Z., *Algebra i geometria analityczna. Definicje, twierdzenia, wzory*, Oficyna Wydawnicza GiS, Wrocław 2024.
3. Topp J., *Algebra liniowa*, Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk 2015.
4. Sage Documentation, strona internetowa, https://doc.sagemath.org [dostęp: 18.10.2025].

Spis tabel

Spis rysunków

Streszczenie

Książka stanowi pierwszą część serii poświęconej algebrze w systemie **SageMath** i obejmuje zagadnienia związane z liczbami zespolonymi. Na początku przedstawiono sposób korzystania z platformy **CoCalc** oraz notatników **Jupyter**, co pozwala szybko zapoznać się ze środowiskiem pracy w **SageMath**. Następnie zaprezentowano podstawowe elementy składni języka **Sage** wraz z najczęściej popełnianymi błędami. Szczególny nacisk położono na definiowanie funkcji, tworzenie rysunków oraz rozwiązywanie równań. Wszystkie omawiane pojęcia i techniki zilustrowano licznymi przykładami, które umożliwiają stopniowe przyswajanie materiału i samodzielne poszukiwanie nowych rozwiązań.

Kolejne rozdziały wprowadzają podstawową teorię dotyczącą liczb zespolonych. Omówiono w nich także zagadnienia związane z zaznaczaniem obszarów na płaszczyźnie zespolonej, pierwiastkowaniem w ciele liczb zespolonych oraz rozwiązywaniem równań zespolonych. Rysunki ilustrujące omawiane pojęcia zostały wykonane w **SageMath**, a przy każdym z nich zamieszczono odpowiedni kod źródłowy. Materiał przedstawiono za pomocą wielu praktycznych przykładów, które prezentują zarówno sposób ręcznego rozwiązywania konkretnych zadań, jak i ich implementację w języku **Sage**.

Abstract

This book is the first in a series on algebra in **SageMath** and focuses on topics related to complex numbers. It begins with an introduction to the **CoCalc** platform and **Jupyter** notebooks, allowing readers to quickly become familiar with the **SageMath** working environment. The book then presents the basic elements of **Sage** syntax, along with common errors. Particular emphasis is placed on defining functions, creating diagrams, and solving equations. All concepts and techniques are illustrated with numerous examples, enabling gradual learning and independent development of new solutions.

Subsequent chapters introduce the fundamental theory of complex numbers. They also cover topics such as plotting regions in the complex plane, extracting roots in the field of complex numbers, and solving complex equations. The diagrams illustrating the concepts are created in **SageMath**, with each accompanied by the corresponding source code. The material is presented through practical examples that demonstrate both manual solutions to specific problems and their implementation in **Sage**.

Politechnika
Białostocka