Algorytmy Geometryczne - Laboratorium 1

Badanie dokładności predykatów geometrycznych

Imię i nazwisko: Wermiński Gabriel **Data wykonania:** 21.10.2025

Grupa: 6

1. Środowisko obliczeniowe

Eksperymenty zostały przeprowadzone w następującej konfiguracji:

- **Procesor:** CPU 12th Gen Intel(R) Core(TM) i5-1235U, 1.30 GHz
- **Pamięć RAM:** 16.0 GB, 3200 MT/s
- **System operacyjny:** Microsoft Windows 11 Home
- **Środowisko programistyczne:** Jupyter Notebook, Pycharm
- **Język programowania:** Python 3.13.5
- **Biblioteki:** numpy, pandas, matplotlib, pathlib

2. Wprowadzenie i cel

Celem laboratorium jest zbadanie dokładności podstawowych operacji geometrycznych, w szczególności predykatu określającego położenie punktu względem prostej. Analiza skupia się na wpływie różnych czynników na poprawność klasyfikacji punktów:

- metody obliczania wyznacznika (implementacja własna vs biblioteczna),
- rodzaju wyznacznika (macierz 2×2 vs 3×3),
- precyzji reprezentacji liczb zmiennoprzecinkowych,
- wartości tolerancji dla zera.

Predykat geometryczny oparty jest na obliczeniu wyznacznika macierzy. Dla trzech punktów a, b i c wartość wyznacznika określa, po której stronie prostej przechodzącej przez punkty a i b znajduje się punkt c.

3. Podstawy teoretyczne

3.1. Orientacja punktu względem prostej

Położenie punktu $c = \left(c_x, c_y\right)$ względem prostej wyznaczonej przez punkty $a = \left(a_x, a_y\right)$ i $b = \left(b_x, b_y\right)$ można określić obliczając wyznacznik:

Macierz 2×2:

$$\det_{2x2(a,b,c)} = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}$$

Macierz 3×3:

$$\det_{3x3(a,b,c)} = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

Interpretacja wyniku:
- $\det > 0 \rightarrow$ punkt leży po lewej stronie prostej,
- $\det = 0 \rightarrow$ punkt leży na prostej,
- $\det < 0 \rightarrow$ punkt leży po prawej stronie prostej.

W praktyce, ze względu na błędy zaokrągleń w arytmetyce zmiennoprzecinkowej, wprowadza się tolerancję ε. Punkt uznaje się za leżący na prostej, gdy $|\det| \leq \varepsilon$.

3.2. Problematyka numeryczna

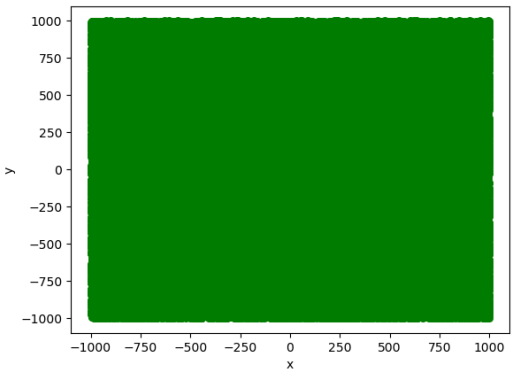
Reprezentacja liczb zmiennoprzecinkowych w komputerze jest ograniczona, co prowadzi do błędów zaokrągleń. Dla dużych wartości liczbowych dokładność względna maleje, ponieważ stała liczba bitów mantysy musi reprezentować coraz większe odległości między kolejnymi liczbami. Operacje na dużych, podobnych liczbach (np. odejmowanie) są szczególnie podatne na utratę cyfr znaczących.

4. Realizacja eksperymentu

4.1. Generowanie zbiorów testowych

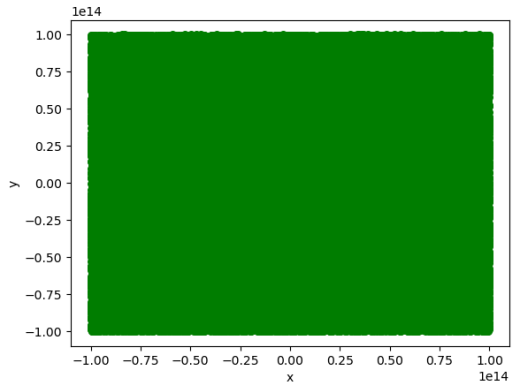
Przygotowano cztery zbiory punktów o różnych charakterystykach:

Zbiór A: Zawiera 10^5 punktów losowo rozłożonych w kwadracie o współrzędnych z przedziału $[-1000, 1000]^2$. Jest to zbiór o umiarkowanym zakresie wartości.



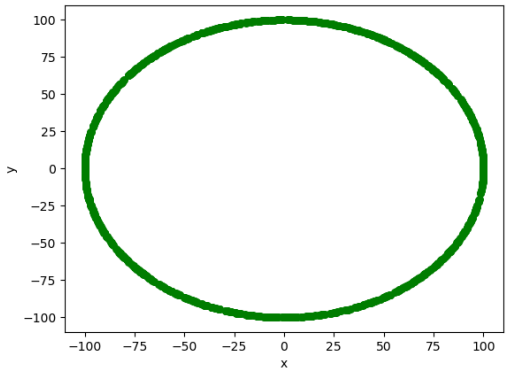
Rysunek 1: Wizualizacja zbioru A

Zbiór B: Obejmuje 10^5 punktów w znacznie większym obszarze: $\left[-10^{14}, 10^{14}\right]^2$. Duże wartości współrzędnych testują stabilność numeryczną algorytmów.



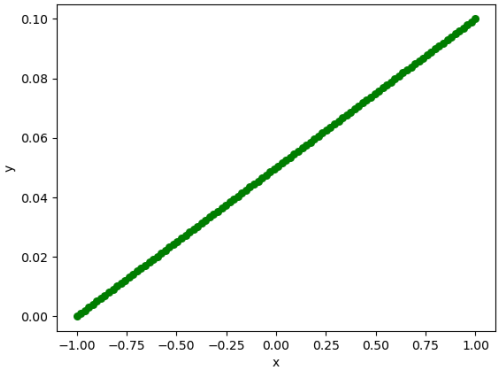
Rysunek 2: Wizualizacja zbioru B

Zbiór C: Składa się z 1000 punktów równomiernie rozłożonych na okręgu o środku w początku układu współrzędnych i promieniu $R = 100$. Punkty generowano w układzie biegunowym, a następnie przekształcano do kartezjańskiego.



Rysunek 3: Wizualizacja zbioru C

Zbiór D: Zawiera 1000 punktów leżących na prostej określonej przez punkty $a = (-1.0, 0.0)$ i $b = (1.0, 0.1)$. Punkty generowano parametrycznie jako kombinacje liniowe końców odcinka, co pozwala zbadać zachowanie predykatu w przypadku idealnym.



Rysunek 4: Wizualizacja zbioru D

4.2. Implementacja wyznaczników

Zaimplementowano cztery warianty obliczania wyznacznika:

Własna implementacja 2×2: Bezpośrednie obliczenie według wzoru $(a_x - c_x)\big(b_y - c_y\big) - \big(a_y - c_y\big)(b_x - c_x)$.

Biblioteczna implementacja 2×2: Wykorzystanie funkcji `numpy.linalg.det()` dla macierzy 2×2.

Własna implementacja 3×3: Upraszczając tę macierz przez odjęcie drugiego wiersza od trzeciego i odjęcie pierwszego wiersza od drugiego otrzymamy:

$$\det(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x - a_x & b_y - a_y & 0 \\ c_x - b_x & c_y - b_y & 0 \end{vmatrix} = (b_x - a_x)\big(c_y - b_y\big) - \big(b_y - a_y\big)(c_x - b_x)$$

Biblioteczna implementacja 3×3: Funkcja `numpy.linalg.det()` dla macierzy 3×3.

4.3. Parametry badania

Klasyfikację punktów przeprowadzono dla następujących konfiguracji:

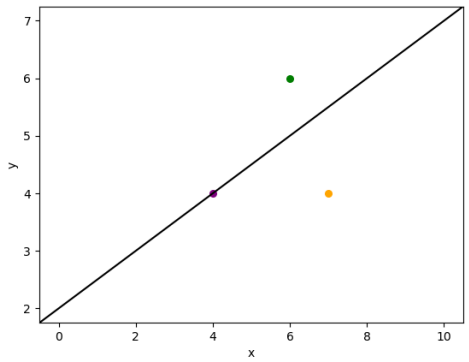
- **Precyzja zmiennych:** `float64` (podwójna) i `float32` (pojedyncza),
- **Tolerancja dla zera:** $\varepsilon \in \{0, 10^{-14}, 10^{-12}, 10^{-10}, 10^{-8}\}$,
- **Metody obliczania:** wszystkie cztery warianty wyznacznika.

Dla każdej konfiguracji rejestrowano liczbę punktów sklasyfikowanych jako leżące po lewej stronie, na prostej oraz po prawej stronie.

4.4. Klasyfikacja punktów względem prostej

Na wykresie przedstawiono przykład klasyfikacji punktów względem prostej (czarna linia):

- Punkty po lewej stronie prostej są oznaczone kolorem zielonym,
- Punkty leżące dokładnie na prostej – kolorem fioletowym,
- Punkty po prawej stronie prostej – kolorem pomarańczowym.



Rysunek 5: klasyfikacja punktów

5. Wyniki i analiza

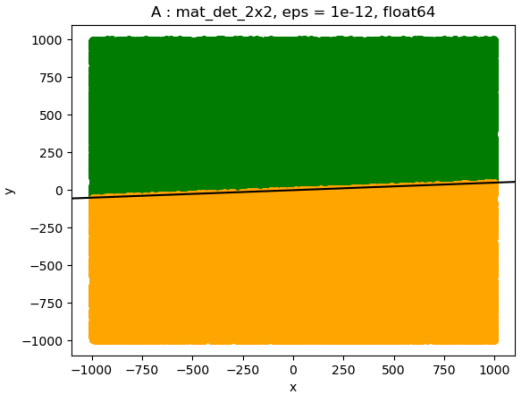
5.1. Zbiór A – zakres umiarkowany (float64)

Metoda	Lewo	Na prostej	Prawo
Wszystkie	50261	0	49739

Tabela 1: Klasyfikacja punktów zbioru A

Dla zbioru A wszystkie konfiguracje (precyzja, tolerancja ε, metoda wyznacznika) dały identyczne wyniki. Rozkład jest niemal równomierny względem prostej. Żaden punkt nie został sklasyfikowany jako leżący na prostej, co jest zgodne z oczekiwaniami.

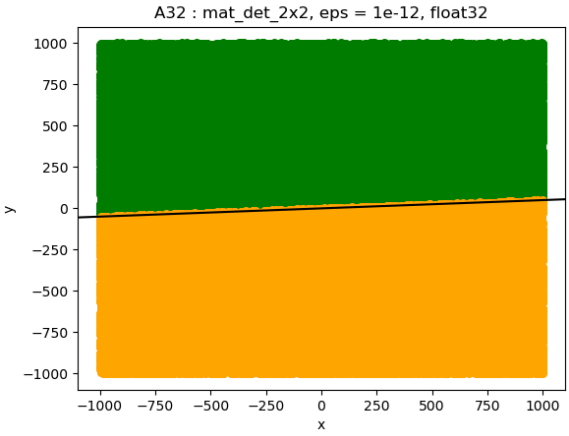
Wniosek: Dla współrzędnych z przedziału $[-1000, 1000]$ wszystkie metody są wystarczająco dokładne.



Rysunek 6: Rozkład punktów zbioru A

5.2. Zbiór A32 – zakres umiarkowany (float32)

Wyniki dla float32 są identyczne jak dla float64: 50261 punktów po lewej, 0 na prostej, 49739 po prawej. Mniejsza precyzja nie wpływa na klasyfikację przy tym zakresie wartości, co potwierdza wystarczającą dokładność float32 dla umiarkowanych współrzędnych.



Rysunek 7: Rozkład punktów zbioru A32

5.3. Zbiór B – duże wartości (float64)

ε	Metoda	Lewo	Na prostej	Prawo
0 do 10^{-8}	mat_det_2x2	49932	4	50064
0 do 10^{-8}	mat_det_2x2_lib	49935	0	50065
0 do 10^{-8}	Wszystkie inne	49934	0	50066

Tabela 2: Klasyfikacja dla zbioru B

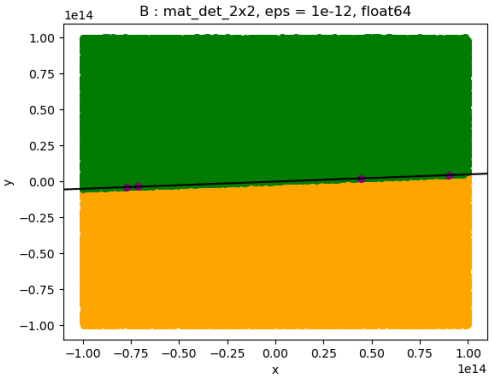
Obserwacje: Funkcja `mat_det_2x2` jako jedyna klasyfikuje 4 punkty jako leżące „na prostej" dla zakresu ε od 0 do 10^{-8}. Pozostałe metody nie wskazują żadnych punktów na prostej. Wyniki są stabilne dla różnych wartości tolerancji.

Analiza sklasyfikowanych punktów: Współrzędne czterech punktów sklasyfikowanych jako leżące na prostej przez `mat_det_2x2`:

- $(4.43 \times 10^{13}, 2.22 \times 10^{12})$
- $(-7.14 \times 10^{13}, -3.58 \times 10^{12})$
- $(-7.72 \times 10^{13}, -3.85 \times 10^{12})$
- $(8.99 \times 10^{13}, 4.47 \times 10^{12})$

Wszystkie punkty mają współrzędne rzędu 10^{13}-10^{14}, co przy precyzji float64 (15-16 cyfr znaczących) znajduje się w granicznym zakresie dokładności reprezentacji.

Interpretacja: Przy wartościach rzędu 10^{14} float64 zapewnia wystarczającą precyzję, ale zbliża się do granicy możliwości. Fakt, że tylko `mat_det_2x2` wykryła te punkty (podczas gdy inne metody nie) sugeruje, że punkty znajdują się bardzo blisko prostej, a różnice w wynikach wynikają z odmiennej propagacji błędów zaokrągleń w różnych implementacjach. Prawdopodobieństwo, że losowe punkty z tak dużego przedziału trafią dokładnie na prostą jest znikome, więc te 4 punkty to najprawdopodobniej artefakt numeryczny.

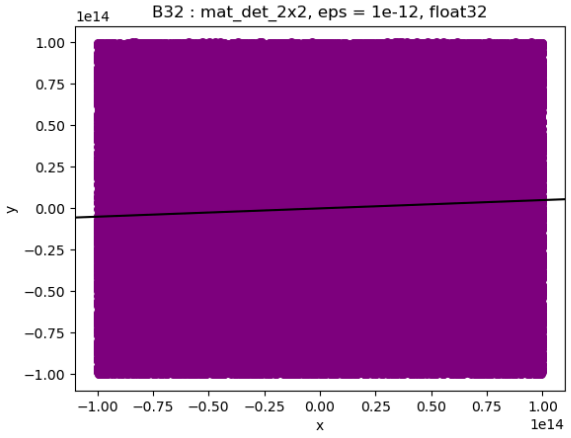


Rysunek 8: Rozkład punktów zbioru B

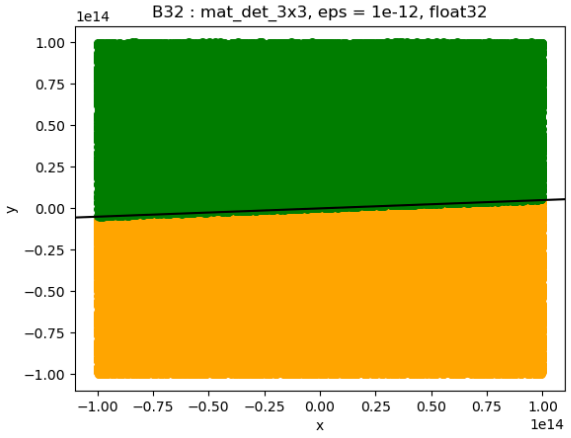
5.4. Zbiór B32 – duże wartości (float32)

ε	Metoda	Lewo	Na prostej	Prawo
10^{-8} do 0	mat_det_2x2	0	100000	0
10^{-8} do 0	mat_det_2x2_lib	0	100000	0
10^{-8} do 0	mat_det_3x3	49934	0	50066
10^{-8} do 0	mat_det_3x3_lib	49934	0	50066

Tabela 3: Klasyfikacja dla zbioru B32



Rysunek 9: B32: det_2x2 (100% błąd)



Rysunek 10: B32: det_3x3 (poprawna klasyfikacja)

Obserwacje krytyczne: Obie implementacje wyznacznika 2×2 (`mat_det_2x2` i `mat_det_2x2_lib`) całkowicie zawiodły – wszystkie 100000 punktów zostały błędnie sklasyfikowane jako leżące na prostej. Jest to 100% błąd klasyfikacji. Natomiast obie implementacje wyznacznika 3×3 zachowują pełną poprawność, dając wyniki niemal identyczne jak dla float64 (różnica w rozkładzie lewo/prawo: 49934 vs 50066).

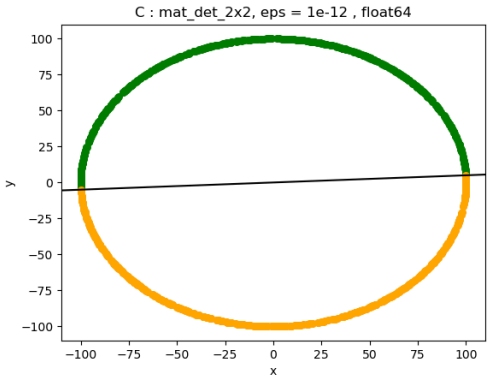
Analiza przyczyn: Float32 oferuje jedynie 7 cyfr znaczących, co przy wartościach rzędu 10^{14} oznacza, że najmniejsza reprezentowalna różnica między liczbami wynosi około 10^7. W wyznacznikach 2×2 obliczamy różnice współrzędnych typu $(b_x - a_x)$ i $(c_x - b_x)$, gdzie wszystkie wartości są rzędu 10^{13}-10^{14}. Odejmowanie tak dużych, losowych liczb powinno dawać wyniki rzędu 10^{13}, ale ograniczona mantysa float32 powoduje, że wynik jest zaokrąglany tak drastycznie, iż ostateczna wartość wyznacznika zawsze wynosi zero (lub wartość poniżej tolerancji).

Dlaczego 3×3 działa lepiej? Metoda 2×2 oblicza wszystkie różnice współrzędnych względem punktu c, podczas gdy metoda 3×3 (po uproszczeniu macierzy) oblicza różnice między kolejnymi punktami. Ta zmiana par odejmowanych punktów prowadzi do innego profilu błędów numerycznych. Przy float32 i wartościach 10^{14} różnice względem wspólnego punktu tworzą korelację prowadzącą do zerowego wyniku, podczas gdy różnice między kolejnymi punktami są bardziej odporne na katastrofalną utratę precyzji.

5.5. Zbiór C – punkty na okręgu (float64)

Wszystkie konfiguracje dały identyczne wyniki: 516 punktów po lewej i 484 po prawej stronie prostej. Żaden punkt nie został sklasyfikowany jako leżący na prostej.

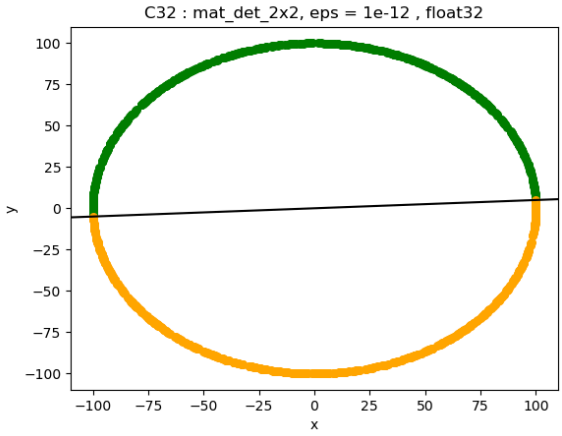
Interpretacja: Równomierny rozkład na okręgu z zakresem $|x|, |y| \leq 100$ zapewnia pełną stabilność obliczeń niezależnie od metody i precyzji.



Rysunek 11: Rozkład punktów zbioru C

5.6. Zbiór C32 – punkty na okręgu (float32)

Wyniki identyczne jak dla zbioru C (float64): 516 po lewej, 0 na prostej, 484 po prawej. Zmniejszona precyzja nie wpływa na klasyfikację przy tym zakresie współrzędnych, co potwierdza stabilność algorytmów dla punktów na okręgu.

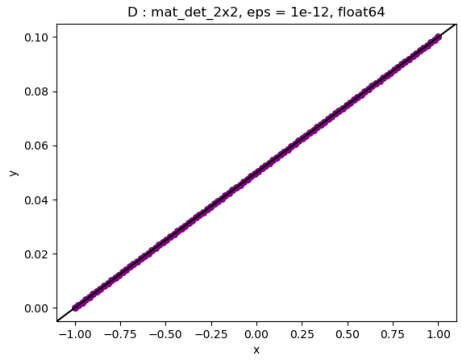


Rysunek 12: Rozkład punktów zbioru C32

5.7. Zbiór D – punkty na prostej (float64)

ε	**Metoda**	**Lewo**	**Na prostej**	**Prawo**
0	mat_det_2x2	340	303	357
0	mat_det_2x2_lib	490	4	506
0	mat_det_3x3	199	591	210
0	mat_det_3x3_lib	164	723	113
10^{-14} do 10^{-8}	wszystkie	0	1000	0

Tabela 4: Wyniki klasyfikacji dla zbioru D w zależności od ε i metody



Rysunek 13: Wizualizacja zbioru D dla $\varepsilon = 10^{-12}$ – wszystkie punkty (kolor fioletowy) poprawnie sklasyfikowane jako leżące na prostej

Obserwacje dla $\varepsilon \in [10^{-14}, 10^{-8}]$: Wszystkie metody osiągają 100% dokładność (1000/1000 punktów na prostej). Wizualizacja potwierdza – wszystkie punkty są fioletowe, bez błędnych klasyfikacji.

Obserwacje dla $\varepsilon = 0$: Wyniki dramatycznie się pogarszają:

- `mat_det_3x3_lib`: 72% poprawnie
- `mat_det_3x3`: 59% poprawnie
- `mat_det_2x2`: 30% poprawnie
- `mat_det_2x2_lib`: tylko 0,4% poprawnie

Błędnie sklasyfikowane punkty dzielą się niemal równo na lewą i prawą stronę, co wskazuje na losowy charakter błędów numerycznych.

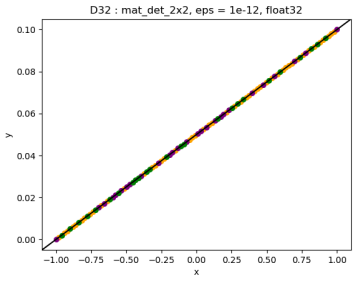
Interpretacja: Punkty wygenerowane parametrycznie powinny teoretycznie leżeć dokładnie na prostej, ale błędy zaokrągleń podczas generacji i obliczeń sprawiają, że wyznacznik jest bliski zeru, lecz niezerowy. Wyznaczniki 3×3 są bardziej stabilne niż 2×2, gdyż używają bezpośrednio współrzędnych zamiast ich różnic, co redukuje utratę cyfr znaczących. Katastrofalny wynik `mat_det_2x2_lib` (0,4%) sugeruje szczególnie niekorzystną propagację błędów w tej implementacji.

Wartość $\varepsilon = 10^{-12}$ jest optymalnym kompromisem – zapewnia pełną dokładność przy dwurzędowym marginesie bezpieczeństwa względem rzeczywistych błędów (10^{-14}).

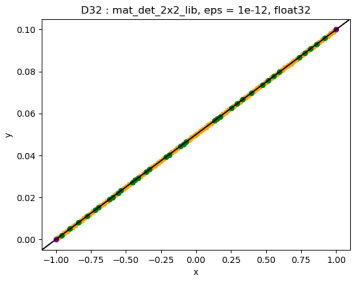
5.8. Zbiór D32 – punkty na prostej (float32)

ε	Metoda	Lewo	Na prostej	Prawo
0 do 10^{-8}	mat_det_2x2	172	656	172
0 do 10^{-8}	mat_det_2x2_lib	500	0	500
0 do 10^{-8}	mat_det_3x3	233	534	233
0 do 10^{-8}	mat_det_3x3_lib	469	62	469

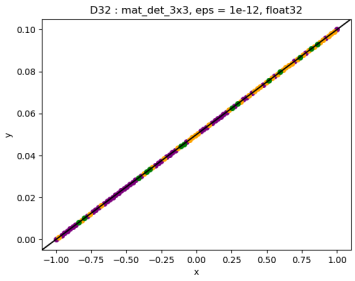
Tabela 5: Wyniki dla zbioru D32



Rysunek 14: D32: det_2x2



Rysunek 15: D32: det_2x2_lib



Rysunek 16: D32: det_3x3

Obserwacje: Dla float32 żadna metoda nie osiąga pełnej dokładności. Najlepsza (`mat_det_2x2`) klasyfikuje 65,6% poprawnie. Biblioteczna `mat_det_2x2_lib` w ogóle nie wykrywa punktów na prostej (0%). Wyznacznik 3×3 osiąga 53,4% dokładności. Wyniki są stabilne dla różnych ε, co sugeruje, że problem leży w ograniczonej precyzji, a nie w tolerancji.

Interpretacja: Float32 jest niewystarczający nawet dla umiarkowanego zakresu współrzędnych, gdy wymagana jest wysoka dokładność klasyfikacji punktów leżących na prostej. Własne implementacje radzą sobie lepiej niż biblioteczne, ale wciąż daleko od idealnych wyników float64.

6. Podsumowanie i wnioski

Na podstawie przeprowadzonych eksperymentów dotyczących klasyfikacji punktów względem prostej przy użyciu różnych metod obliczania wyznaczników można sformułować następujące wnioski:

1. **Zakres wartości współrzędnych ma kluczowe znaczenie dla stabilności numerycznej.** Dla zbiorów A i C (współrzędne rzędu setek lub tysięcy) wszystkie badane metody dały identyczne i poprawne wyniki, niezależnie od precyzji i tolerancji. Problemy numeryczne ujawniają się dopiero przy wartościach ekstremalnych (zbiór B: 10^{14}).
2. **Float32 jest całkowicie nieadekwatny dla dużych wartości liczbowych.** W zbiorze B32 obie implementacje wyznacznika 2×2 (`mat_det_2x2` i `mat_det_2x2_lib`) całkowicie zawiodły, błędnie klasyfikując wszystkie 100000 punktów jako leżące na prostej (100% błąd). Tylko wyznaczniki 3×3 zachowały poprawność, dając wyniki identyczne jak dla float64.

3. **Wyznaczniki 3×3 oferują znacznie lepszą stabilność numeryczną niż 2×2** w ekstremalnych warunkach. Mimo większej liczby operacji arytmetycznych, bezpośrednie użycie współrzędnych punktów (zamiast ich różnic) zapewnia lepszą odporność na błędy zaokrągleń. Pokazuje to, że w arytmetyce zmiennoprzecinkowej „więcej operacji" nie zawsze oznacza „większy błąd" – liczy się struktura obliczeń.

4. **Tolerancja $\varepsilon = 10^{-12}$ jest optymalnym wyborem dla float64.** Dla zbioru D (punkty na prostej) wartość z zakresu $[10^{-14}, 10^{-8}]$ zapewniła 100% dokładność klasyfikacji dla wszystkich metod. Bez tolerancji ($\varepsilon = 0$) wyniki dramatycznie spadły: od 72% dla `mat_det_3x3_lib` do zaledwie 0,4% dla `mat_det_2x2_lib`.

5. **Własne implementacje mogą znacząco przewyższać funkcje biblioteczne.** Najwyraźniej widać to w zbiorze D przy $\varepsilon = 0$: własna implementacja `mat_det_3x3` osiągnęła 59% dokładności vs 72% bibliotecznej, ale własna `mat_det_2x2` osiągnęła 30% vs katastrofalne 0,4% bibliotecznej. Dla zbioru D32 różnice są jeszcze większe: własna `mat_det_2x2` uzyskała 65,6% dokładności, podczas gdy biblioteczna w ogóle nie wykryła punktów na prostej (0%).

6. **Precyzja float64 jest niezbędna dla zadań wymagających wysokiej dokładności.** Zbiory A32 i C32 pokazały, że float32 wystarcza dla prostych przypadków (identyczne wyniki jak float64), ale zbiory B32 i D32 wykazały jego całkowitą niewystarczalność w scenariuszach wymagających – błędy klasyfikacji sięgają 100% dla dużych wartości i 35-100% dla punktów na prostej.

7. **Klasyfikacja punktów leżących dokładnie na prostej jest szczególnie trudna numerycznie.** Nawet punkty wygenerowane parametrycznie jako kombinacje liniowe (zbiór D) nie dają wyznacznika równego dokładnie zeru. Błędy zaokrągleń podczas generacji i obliczeń są nieuniknione. Właściwy dobór tolerancji ε jest absolutnie kluczowy – bez niej nawet najlepsza metoda osiąga maksymalnie 72% dokładności.

8. **Wyniki są stabilne względem tolerancji w szerokim zakresie** (10^{-8} do 10^{-14}) dla zbiorów A, B i C. Wrażliwość na wartość ε pojawia się tylko dla zbioru D, gdzie punkty rzeczywiście leżą na prostej. To potwierdza, że dla typowych zastosowań geometrycznych (punkty nieleżące na prostej) wybór konkretnej wartości tolerancji ma niewielkie znaczenie.

9. **Artefakty numeryczne są możliwe nawet dla float64.** W zbiorze B funkcja `mat_det_2x2` sklasyfikowała 4 losowe punkty jako leżące „na prostej", mimo że prawdopodobieństwo takiego zdarzenia w przedziale 10^{14} jest praktycznie zerowe. To pokazuje, że nawet przy wystarczającej precyzji błędy zaokrągleń mogą prowadzić do fałszywych klasyfikacji.

10. **Wizualizacja graficzna jest cennym narzędziem weryfikacji.** Wykresy dla $\varepsilon = 10^{-12}$ wyraźnie pokazują poprawność klasyfikacji (wszystkie punkty fioletowe dla zbioru D w float64) oraz dramatyczne różnice między metodami (100% błąd dla B32 z metodą 2×2).

6.1. Rekomendacje praktyczne

Dla zastosowań praktycznych w algorytmach geometrycznych zaleca się:

- Obligatoryjne używanie **float64** jako standardu. Float32 może być stosowany wyłącznie dla niewielkich zakresów wartości ($|x|, |y| < 1000$) i gdy nie wymaga się wysokiej dokładności.
- Preferowanie **wyznacznika 3×3** jako uniwersalnej metody zapewniającej najlepszą stabilność numeryczną, szczególnie dla nieznanych z góry zakresów współrzędnych lub gdy możliwe są duże wartości.
- Ustawienie **tolerancji $\varepsilon = 10^{-12}$** dla float64 jako bezpiecznego standardu – zapewnia pełną dokładność z dwurzędowym marginesem bezpieczeństwa względem rzeczywistych błędów (10^{-14}).

- **Bezwzględne stosowanie tolerancji numerycznej** – brak tolerancji ($\varepsilon = 0$) prowadzi do 28-99,6% błędów nawet dla idealnych danych teoretycznych.
- **Testowanie algorytmów** na różnorodnych zbiorach danych, w tym przypadkach ekstremalnych: duże wartości współrzędnych (10^{14}), punkty leżące na testowanych obiektach geometrycznych, różne precyzje arytmetyczne.
- Jeśli wydajność jest krytyczna i zakres współrzędnych jest niewielki, można użyć **wyznacznika 2×2 z float64**, który wymaga mniej operacji przy zachowaniu dokładności.
- W przypadku wdrażania własnych implementacji – dokładne testowanie i porównanie z funkcjami bibliotecznymi, gdyż mogą one dawać różne wyniki (lepsze lub gorsze) w zależności od przypadku użycia.