Sprawozdanie z laboratorium 1 z algorytmów geometrycznych

Hubert Miklas
Grupa 5, Poniedziałek 9:45-11:15
Data wykonania ćwiczenia:

Data oddania sprawozdania:

October 28, 2024

1 Wstęp do ćwiczenia

1.1 Specyfikacja środkowiska

System: Ubuntu 20.04.6 LTS x86_64
Procesor: Intel i5-6400 (4) @ 3.300GHz
Pamięć RAM: 16 GB
Karta Graficzna: NVIDIA GeForce GTX 970
Środowisko: Jupyter Notebook, Python 3.8.10

1.2 Opis i cel ćwiczenia

Celem ćwiczenia jest wygenerowanie różnych zbiorów punktów na płaszczyźnie oraz dokonanie ich podziału według orientacji względem zadanej prostej, przechodzącej przez punkty $a = (-1, 0)$ i $b = (1, 0.1)$. W ramach analizy wykorzystuję samodzielne zaimplementowane wyznaczniki przedstawione w opisie ćwiczenia, jak i funkcji z biblioteki numpy: numpy.linalg.det(numpy.matrix()) do wyznaczania orientacji punktów względem zdanej prostej. Dokonuję także porównania wpływu tolerancji oraz precyzji obliczeń na klasyfikację punktów i różnicę czasu obliczeń punktów.

1.3 Wstęp teoretyczny

Poniżej zamieszczam wzory wyznaczników wykorzystanych do wyznaczenia podanych poniżej danych.

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

(a) Wyznacznik 3x3 dla punktów a,b,c

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

(b) Wyznacznik 3x3 dla punktów a,b,c

Obraz 1: Wykorzystane wyznaczniki

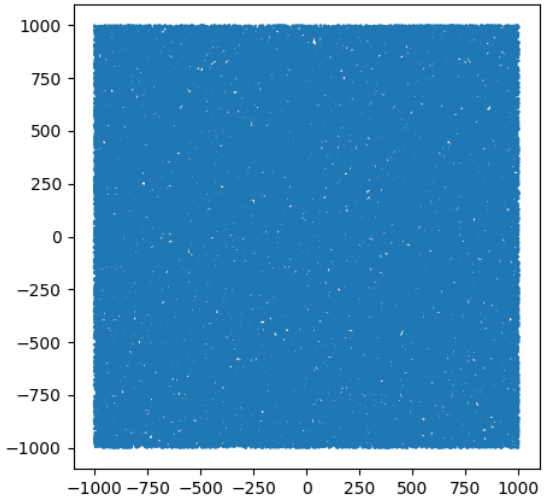
Wyznacznik (b) jest wyprowadzony z wyznacznika (a) poprzez odjęcie trzeciego wiersza od pierwszego i drugiego. Następnie korzystając z rozwinięcia Laplace'a na ostatnim rzędzie, otrzymujemy (b).

2 Generowanie danych

Poniżej znajdują się wykresy i opisy metod wykorzystanych do generowania danych.

2.1 Metoda 1: Generowanie zbioru 10^5 losowych punktów w przedziale $[-10^3, 10^3]$

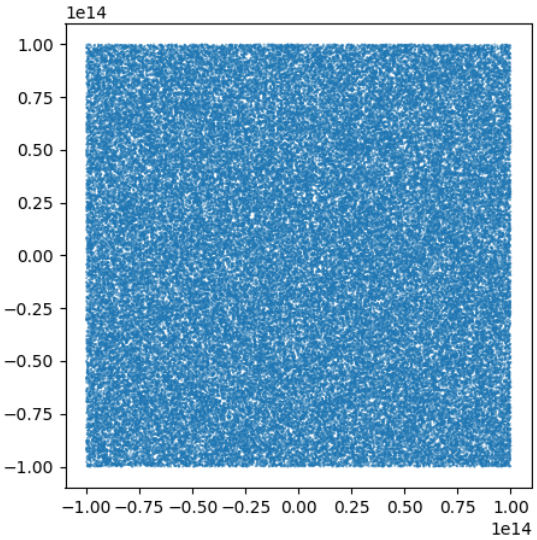
Wygenerowano 10^5 losowych punktów w przedziale $[-1000, 1000]$ przy użyciu funkcji `uniform(-1000, 1000)` z biblioteki `random`. Wynikowa chmura punktów została zwizualizowana na Wykresie 8.



Wykres 2: Wizualizacja losowania zbioru 10^5 punktów na przedziale $[-1000, 1000]$.

2.2 Metoda 2: Generowanie zbioru 10^5 losowych punktów w przedziale $[-10^{14}, 10^{14}]$

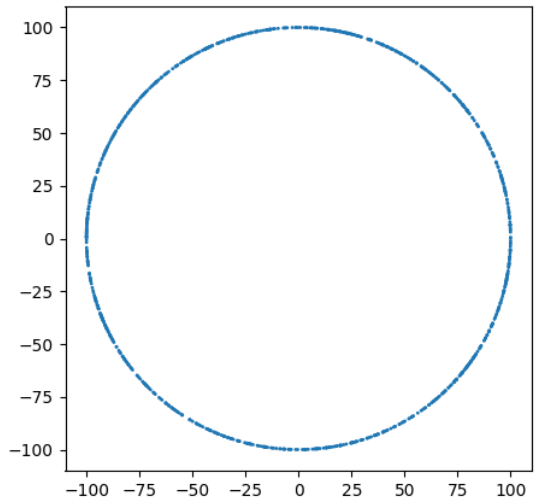
Druga metoda generuje losowe punkty w szerszym zakresie $[-10^{14}, 10^{14}]$. Wynikowa chmura punktów znajduje się na Wykresie 3.



Wykres 3: Wizualizacja losowania 10^5 punktów na przedziale $[-10^{14}, 10^{14}]$.

2.3 Metoda 3: Generowanie zbioru punktów na okręgu o promieniu 100

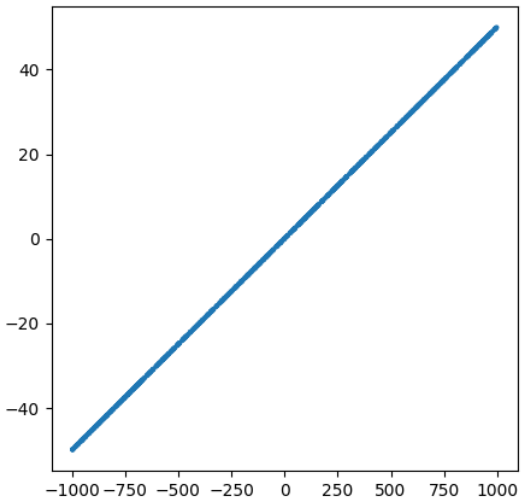
Ta metoda generuje punkty równomiernie rozmieszczone na obwodzie okręgu o promieniu $R = 100$. Wykres tych punktów znajduje się na Wykresie 4.



Wykres 4: Punkty równomiernie rozmieszczone na okręgu o promieniu 100.

2.4 Metoda 4: Generowanie zbioru punktów na prostej

Wygenerowano punkty wzdłuż prostej określonej przez wektory $\mathbf{a} = [-1.0, 0.0]$ oraz $\mathbf{b} = [1.0, 0.1]$. Wykres tych punktów znajduje się na Wykresie 5.



Wykres 5: Punkty rozmieszczone na prostej wyznaczonej przez wektory **a** i **b**.

3 Przebieg obliczeń

Dla każdego zbioru danych przeprowadzono obliczenia w celu podziału punktów względem ich orientacji względem odcinka $\overline{ab}$. Punkty zostały zaklasyfikowane jako:

- Po lewej stronie prostej wyznaczonej przez $\overline{ab}$.
- Po prawej stronie prostej wyznaczonej przez $\overline{ab}$.
- Współliniowe z prostą.

Podział punktów przeprowadzono dwoma metodami:

1. Własnoręcznie zaimplementowane funkcje wyznaczania wyznaczników.
2. Użycie funkcji z biblioteki numpy.

4 Prezentacja wyników

Dla każdego zbioru danych została wykonana funkcja klasyfikująca calculate(), która każdemu punktowi przypisała pozycję względem zadanej prostej. Kody kolorów to: zielony - powyżej prostej, czerwony - poniżej prostej, niebieski - na prostej.

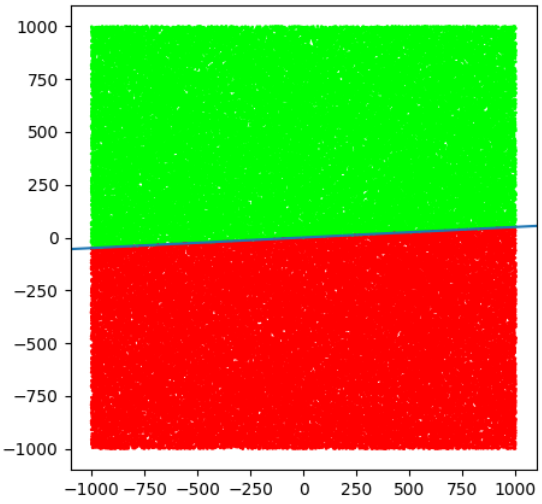
4.1 Zbiór 10^5 losowych punktów w przedziale $[-10^3, 10^3]$

Tabela 1: Tabela różnic czasów obliczeń i tolerancji dla generatora 10^5 z $(-1000, 1000)$

Tolerancja ϵ	Typ numeryczny	Nr. det	Czas obliczeń (s)	Powyżej	Poniżej	Współliniowe
1e-14	float32	1.	6.3011	50188	49812	0
1e-14	float32	2.	4.8937	50188	49812	0
1e-14	float32	3.	6.7522	50188	49812	0
1e-14	float32	4.	6.0813	50188	49812	0

1. - Własna implementacja 3×3
2. - Własna implementacja 2×2
3. - Wyznacznik macierzy 2×2 z numpy
4. - Wyznacznik macierzy 3×3 z numpy

Wyniki nie różnią się znacząco. Czas obliczeń jest najkrótszy dla własnej implementacji wyznacznika 2×2. Jest to prawdopodobnie spowodowane najmniejszą ilością operacji. Poniżej znajduje się wykres obrazujący wynik powyżej:



Wykres 6: Wizualizacja klasyfikacji dla zbioru 10^5 punktów na przedziale $[-1000, 1000]$.

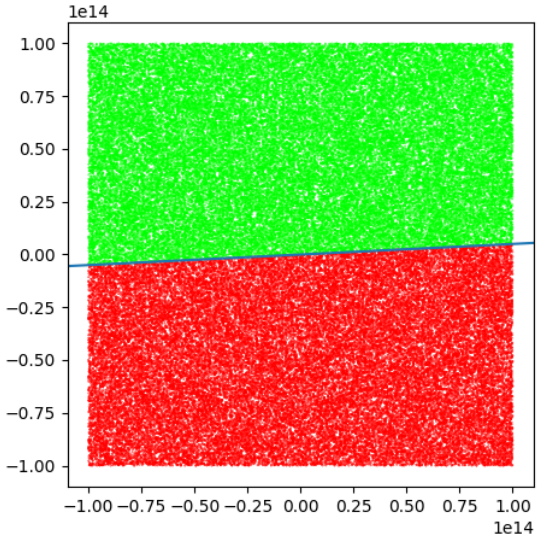
4.2 Zbiór 10^5 losowych punktów w przedziale $[-10^{14}, 10^{14}]$

Tabela 2: Results for generator 10^5 z $(-10^{14}, 10^{14})$

Tolerancja ϵ	Typ numeryczny	Nr. det	Czas obliczeń (s)	Powyżej	Poniżej	Współliniowe
1e-14	float32	1.	6.1396	49958	50042	0
1e-14	float32	2.	4.7918	49958	50042	0
1e-14	float32	3.	6.0965	49958	50042	0
1e-14	float32	4.	5.7973	49958	50042	0

1. - Własna implementacja 3×3
2. - Własna implementacja 2×2
3. - Wyznacznik macierzy 2×2 z numpy
4. - Wyznacznik macierzy 3×3 z numpy

Tu również wyniki nie różnią się znacząco. Czas obliczeń jest dalej najkrótszy dla własnej implementacji wyznacznika 2×2. Poniżej znajduje się wykres obrazujący wynik powyżej:



Wykres 7: Wizualizacja klasyfikacji dla zbioru 10^5 punktów na przedziale $[-10^{14}, 10^{14}]$.

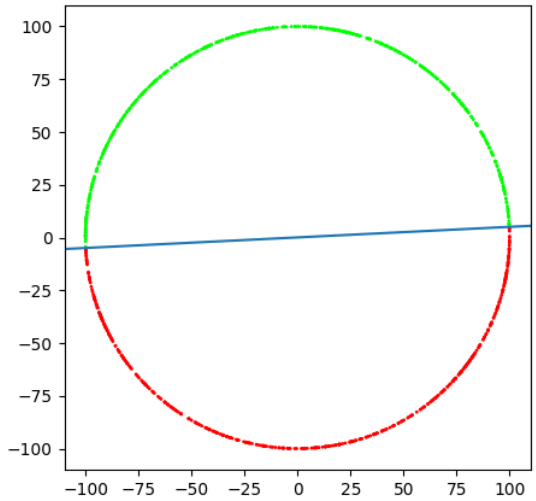
4.3 Okrąg punktów o promieniu 100 i środku $(0, 0)$

Tabela 3: Results for generator Okrąg o promieniu 100 i środku $(0, 0)$

Tolerancja ϵ	Typ numeryczny	Nr. det	Czas obliczeń (s)	Powyżej	Poniżej	Współliniowe
1e-14	float32	1.	0.3573	504	496	0
1e-14	float32	2.	0.2697	504	496	0
1e-14	float32	3.	0.3693	504	496	0
1e-14	float32	4.	0.3275	504	496	0

1. - Własna implementacja 3×3
2. - Własna implementacja 2×2
3. - Wyznacznik macierzy 2×2 z numpy
4. - Wyznacznik macierzy 3×3 z numpy

Również niewielkie różnice. Poniżej znajduje się wykres, który wizualizuje zadane wyniki obliczeń.



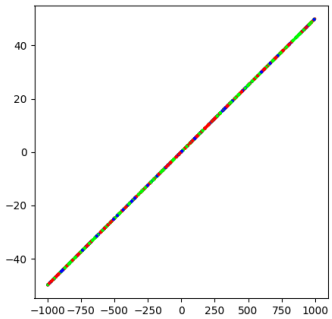
Wykres 8: Wizualizacja klasyfikacji dla zbioru 1000 punktów na przedziale okręgu o promieniu 100 i środku w punkcie (0,0).

4.4 Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$

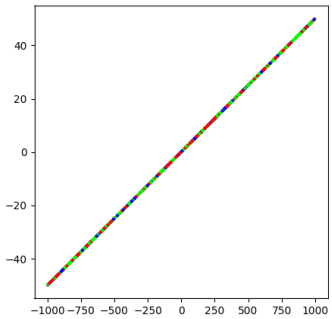
Tabela 4: Tabela różnic czasów obliczeń i tolerancji dla generatora Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$

Tolerancja ϵ	Typ numeryczny	Nr. det	Czas obliczeń (s)	Powyżej	Poniżej	Współliniowe
1e-14	float32	1.	0.5580	425	397	178
1e-14	float32	2.	0.2401	425	418	157
1e-14	float32	3.	0.3327	433	411	156
1e-14	float32	4.	0.4677	433	406	161
1e-08	float32	1.	0.3256	424	395	181
1e-08	float32	2.	0.2762	424	395	181
1e-08	float32	3.	0.4349	424	395	181
1e-08	float32	4.	0.2717	424	395	181
1e-14	float64	1.	0.4033	0	0	1000
1e-14	float64	2.	0.2979	1	220	779
1e-14	float64	3.	0.3168	1	190	809
1e-14	float64	4.	0.2469	17	117	866
1e-08	float64	1.	0.3468	0	0	1000
1e-08	float64	2.	0.2262	0	0	1000
1e-08	float64	3.	0.2651	0	0	1000
1e-08	float64	4.	0.2593	0	0	1000

1. - Własna implementacja 3×3
2. - Własna implementacja 2×2
3. - Wyznacznik macierzy 2×2 z numpy
4. - Wyznacznik macierzy 3×3 z numpy

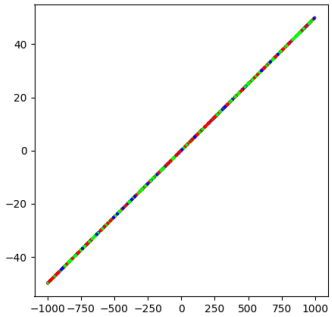


(a) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-8}$, float32, Własna implementacja 2×2

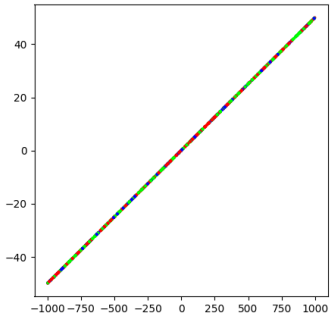


(b) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-8}$, float32, Własna implementacja 3×3

Obraz 9: Wykorzystane wyznaczniki dla $\epsilon = 10^{-8}$ i typu float32

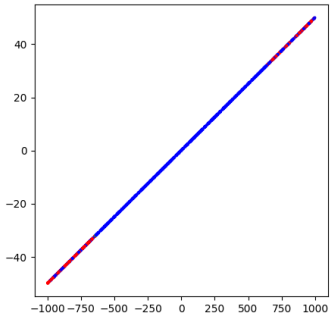


(a) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-8}$, float32, Wyznacznik macierzy 2×2 z numpy

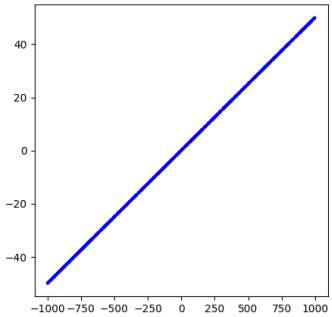


(b) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-8}$, float32, Wyznacznik macierzy 3×3 z numpy

Obraz 10: Wykorzystane wyznaczniki dla $\epsilon = 10^{-8}$ i typu float32, wyznaczniki biblioteczne

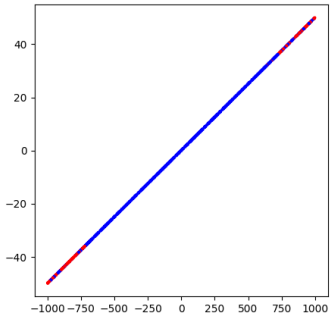


(a) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-14}$, float64, Własna implementacja 2×2

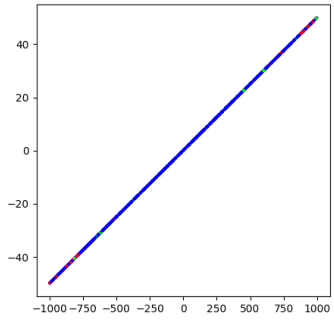


(b) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-14}$, float64, Własna implementacja 3×3

Obraz 11: Wykorzystane wyznaczniki dla $\epsilon = 10^{-14}$ i typu float64



(a) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-14}$, float64, Wyznacznik macierzy 2×2 z numpy



(b) Linia $(x, y) = v \cdot t$, $v = (0, 0.1)$, $t \in \mathbb{R}$, $\epsilon = 10^{-14}$, float64, Wyznacznik macierzy 3×3 z numpy

Obraz 12: Wykorzystane wyznaczniki dla $\epsilon = 10^{-14}$ i typu float64, wyznaczniki biblioteczne

W tym przypadku występują duże różnice. Dłuższą analizę opisałem we wnioskach.

5 Wnioski

Dla zbirów numer 1, 2 i 3 zmiana precyzji obliczeń i dokładności ϵ nie wpływa znacząco na wynik, ponieważ prawdopodobieństwo, że punkt trafi na tę prostą, jest bardzo małe. Dla największego $\epsilon = 10^{-8}$ i zbioru o zakresie $[-1000, 1000] \times [-1000, 1000]$ prawdopodobieństwo to wynosi $L = \{(x, y) \in [-1000, 1000] \times [-1000, 1000] : [0.05x + 0.05 + \epsilon = y, 0.05x + 0.05 + \epsilon = y]\}, \Omega = (x, y) \in [-1000, 1000] \times [-1000, 1000], 10^5 * P((x, y) \in L) = 10^5 \times \frac{10^{-8} \times 2800}{1000 \times 1000} \approx 3 \times 10^{-9}$. W każdym przypadku różnice są poniżej 1% i nie muszą wynikać z dokładności obliczeniowej, a nierównego losowego rozkładu punktów, co jest oczekiwane. W przypadku metody 4. sytuacja jest zgoła inna - wyniki znacząco się różnią. Dla typu float32 i tolerancji $\epsilon \in \{10^{-8}, 10^{-14}\}$zmiany w obliczeniach nie są takie znaczące. Dla wyznacznika własnej implementacji kilkadziesiąt punktów mniej jest klasyfikowane jako współliniowe w porównaniu do tych bibliotecznych. Czasy obliczeń są mocno zróżnicowane i zależą prawdopodobnie od stopnia obciążenia procesorów. Najciekawszy jest przypadek to float64 i $\epsilon = 10^{-14}$. Widać, że najlepiej poradziła sobie własna implementacja wyznacznika 3×3. Druga w kolejności była implementacja wyznacznika bibliotecznego 3×3 błędnie klasyfikując 134 punkty. Pozostałe dwie klasyfikacje dały zbliżony wynik, jednak niewątpliwie funkcja biblioteczna poradziła sobie lepiej. Ciężko jednak wyciągnąć stosowne wnioski, ponieważ traktujemy funkcje biblioteczne jako czarne skrzynki przyjmujące dane i zwracające wynik. Można porównać własne implementacje - wyznacznik 3×3 wykonuje więcej operacji algebraicznych niż 2×2, mimo to daje lepszy wynik. Wydaje się to sprzeczne z intuicją, i prawdopodobnie sprowadza się to do sposobu, w jaki Python implementuje typy zmiennoprzecinkowe.