Diagramy Voronoi

Jarosław Marek Gliwiński
#indeksu 74839

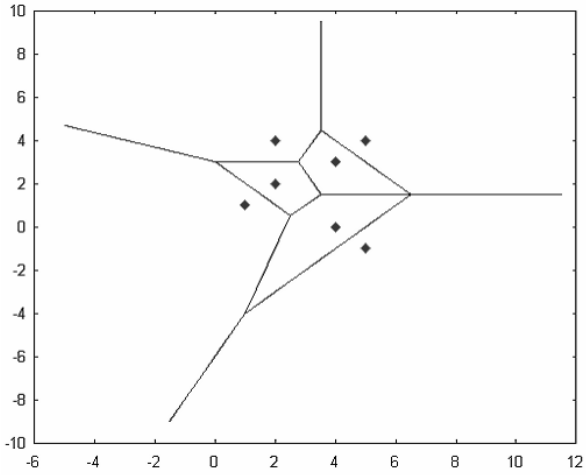
4 października 2009

1 Wstęp

Celem ćwiczenia było zapoznanie z własnościami diagramów Voronoia oraz ich implementacją w środowisku Matlab. Diagramy te, zwane także tesselacjami Dirichleta, w przypadku przestrzeni dwuwymiarowej, dla danego zbioru n punktów, dzielą się płaszczyznę na n obszarów w taki sposób, że każdy punkt w dowolnym obszarze znajduje się bliżej określonego punktu ze zbioru n punktów niż od pozostałych $n-1$ punktów.W skrócie, jest to podział na obszary zawierające punkty, dla których punkt środkowy obszaru jest bliższy niż pozostałe punkty wejściowe.

2 Program „voronoi”

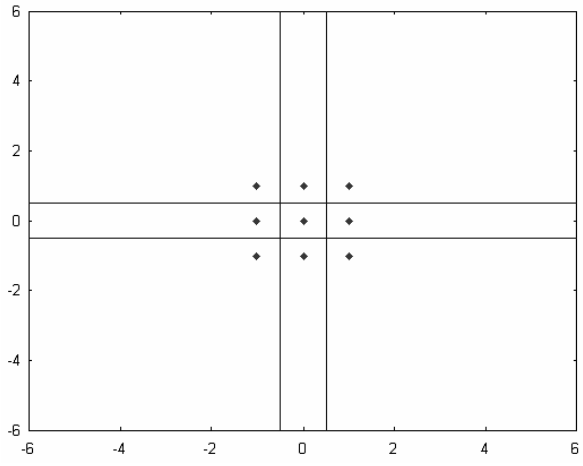
Celem użycia programu była ilustracja działania podziału. Dla przykładowych plików wejściowych uzyskano następujące wyniki (ilustracje 1 – 3 oraz tabele 1–3). Wyniki generowane przez program pokrywały się ze spodziewanymi – a więc były zgodne z wzorcami zamieszczonymi w instrukcji ćwiczenia.



Ryc. 1: Ilustracja działania programu „voronoi”, przypadek pierwszy

x	1	4	2	5	2	2	5	4
y	1	3	2	4	4	2	-1	0

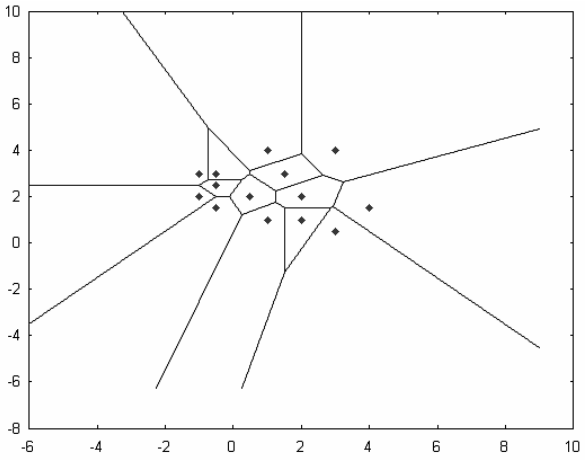
Tabela 1: Dane wejściowe dla przypadku pierwszego



Ryc. 2: Ilustracja działania programu „voronoi", przypadek drugi

x	-1	0	1	-1	0	1	-1	0	1
y	1	1	1	0	0	0	-1	-1	-1

Tabela 2: Dane wejściowe dla przypadku drugiego



Ryc. 3: Ilustracja działania programu „voronoi", przypadek trzeci

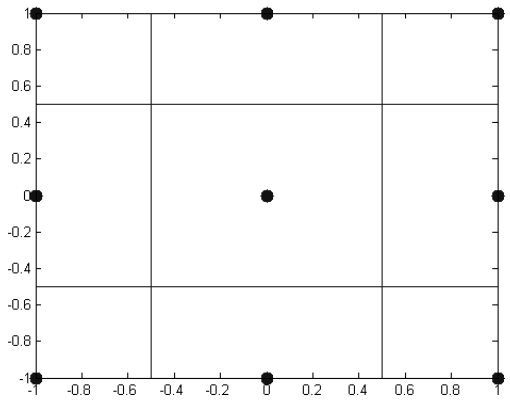
x	1	3	-1	-0.5	1.5	-0.5	-1	0.5	2	-0.5	1	2	4	3
y	4	4	3	3	3	2.5	2	2	2	1.5	1	1	1.5	0.5

Tabela 3: Dane wejściowe dla przypadku trzeciego

3 Skrypt w środowisku Matlab

Implementacji dokonano na podstawie kodu udostępnionego w instrukcji do ćwiczenia. Zamiast wklejać cały skrypt w postaci jak w instrukcji, opisane zostaną zmiany w kodzie prowadzące do uzyskania zadowalającego efektu.

Kod funkcji `[vx,vy] = my_vor(hm,hmt)` okazał się być kompletną implementacją zgodną z opisem teoretycznym, za wyjątkiem, przypadkowego zapewne, błędu polegającego na wywołaniu wbudowanej w główny pakiet Matlaba funkcji wyliczającego triangulację Delaunay – `tri = delaunay(x,y)` poprzedzonej słowem kluczowym języka Matlab służącym do oznaczania definicji funkcji użytkownika – `function`. Po poprawieniu tego prostego błędu (poprzez usunięcie słowa kluczowego, rzecz jasna) skrypt poprawnie generował podział według schematu diagramów Voronoi. Działanie programu po tej poprawce przedstawiono na rysunku 4.



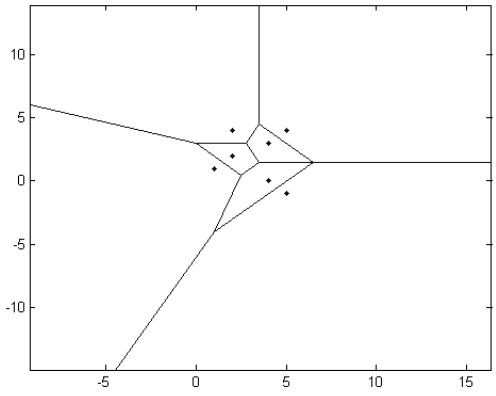
Ryc. 4: Pierwsza wersja skryptu

Zastosowano dwa zabiegi. Po pierwsze, nieprzypadkowo wybrano przykład drugi, na którym łatwo człowiekowi wykonać podział „w pamięci" bądź porównywać wyniki. Po drugie, sztucznie powiększono punkty wejściowe, dzięki czemu widoczne jest, że obszar wyświetlania nie jest dobrany prawidłowo. Automatyczna decyzja w kwestii wyboru obszaru na podstawie skrajnych punktów zbioru wejściowego jest właściwa, jednak należy dodać pewien margines ułatwiający odbiór wyników przez obserwatora. Dobrym rozwiązaniem okazało się być zastosowanie metody wyboru obszaru opartej na końcach (skrajnych wartościach) generowanych przez skrypt linii podziału. Poprawka polegała więc na dopisaniu linii

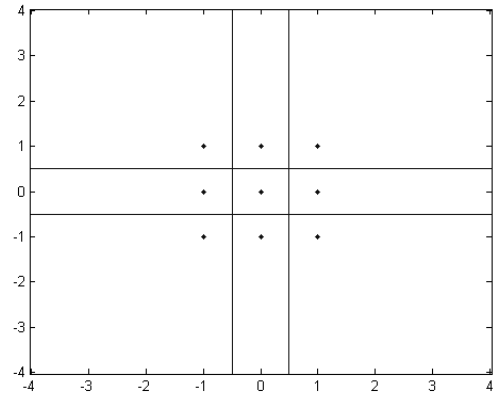
```
axis([min(vx) max(vx) min(vy) max(vy)]);,
```

gdzie `vx` i `vy` to macierze opisujące linie podziału Wyniki tej metody zaprezentowano na rysunkach 5–7.

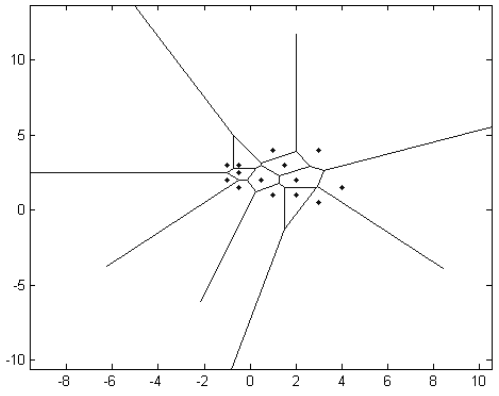
Można uznać, że wyniki drugiej wersji są zadowalające. Jednak sprawdzono jedynie wbudowane przypadki. Po wykonaniu testów dla przypadków losowo generowanych, szczególnie dla większej liczby punktów wejściowych, wyraźnie widać, że generowane linie podziału są zbyt długie, by możliwe było określanie granic wyłącznie na ich podstawie. W związku z tym wprowadzono dodatkowe ograniczenie w postaci porównania wybranych przez drugą wersję ograniczeń z wielokrotnością skrajnych wartości współrzędnych punktów wejściowych. Wybra-



Ryc. 5: Druga wersja skryptu, przypadek pierwszy



Ryc. 6: Druga wersja skryptu, przypadek drugi

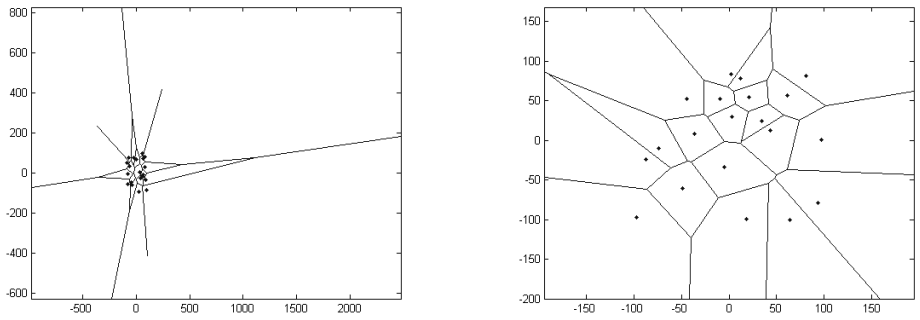


Ryc. 7: Druga wersja skryptu, przypadek trzeci

no $n = 2$, co spowodowało zmianię linii wprowadzonej poprzednio na:

```
   axis([
max([min(min(vx)) n*min(X)]) min([max(max(vx)) n*max(X)])
max([min(min(vy)) n*min(Y)]) min([max(max(vy)) n*max(Y)])
]);
```

Rysunek 8 przedstawia porównanie stanu przed i po korekcie. Rzecz jasna punkty są różne, bowiem wybrano je pseudolosowo, jednak różnica jest wyraźnie widoczna. Dodatkowo poprawiono metodę generacji punktów, tak, by były one z zakresu $< -100100 >$ zamiast $< 0200 >$ – wydawało się to bardziej intuicyjne, biorąc pod uwagę występowanie liczb ujemnych także we wbudowanych przypadkach.



(a) Druga wersja skryptu (b) Trzecia wersja skryptu

Ryc. 8: Wizualizacja podziału dla punktów pseudolosowych