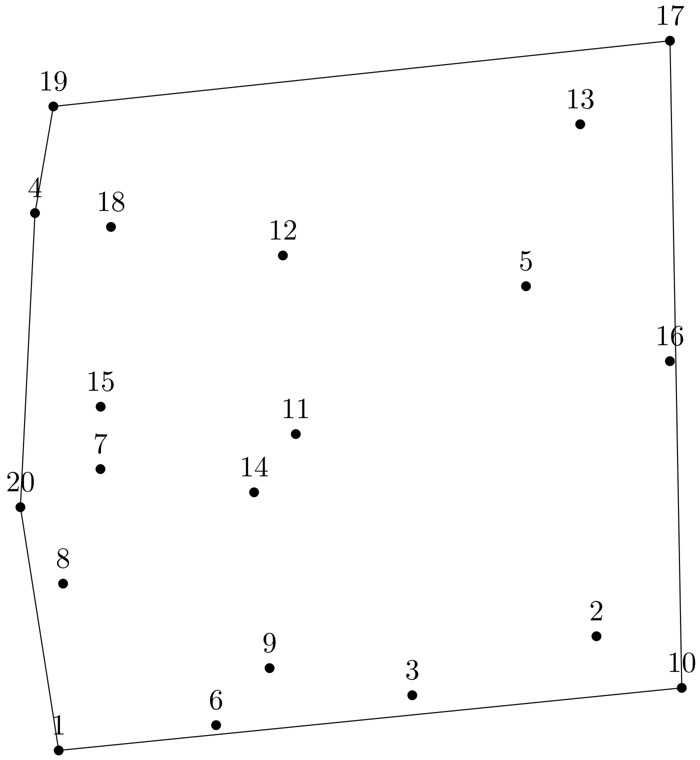
Otoczki wypukłe

Spis treści

Wstęp

Rozdział I
Omówienie teoretyczne otoczki wypukłej na płaszczyźnie

Otoczka wypukła zbioru punktów w swojej najbardziej podstawowej postaci jest wielokątem wypukłym obejmującym wszystkie punkty ze zbioru punktów leżących na płaszczyźnie w taki sposób, aby wielokąt ten miał jak najmniejsze pole.



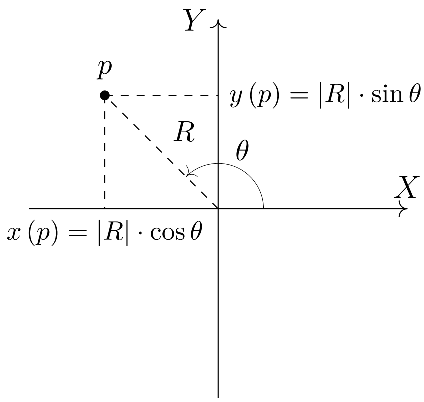
Rysunek I.1: Otoczka wypukła na płaszczyźnie
Źródło: opracowanie własne

Dobrą reprezentacją otoczki wypukłej w świecie fizycznym może być grupa gwoździ przybita do płaskiej powierzchni i następnie opleciona ciasno sznurkiem. Gwoździe stykające się ze sznurkiem stanowić będą wierzchołki otoczki wypukłej tej grupy gwoździ.

1. Otoczka wypukła zbioru punktów

W celu wyznaczenia otoczki wypukłej dla najbardziej ogólnego przypadku — nieuporządkowanego zbioru punktów na płaszczyźnie, możemy wykorzystać dwa najbardziej popularne algorytmy algorytm Jarvisa oraz algorytm Grahama.

W wykorzystywanych algorytmach istotnym elementem jest sortowanie punktów względem wartości, które zostały przedstawione na rysunku I.2. W zależności od tego, w jakiej postaci określone będą dane punkty, należy dokonać odpowiednich obliczeń.



Rysunek I.2: Punkt na układzie współrzędnych
Źródło: opracowanie własne

Gdzie poszczególne symbole oznaczają:

p	-	rozpatrywany punkt
$x(p)$	-	odcięta punktu p
$y(p)$	-	rzędna punktu p
R	-	długość wektora wodzącego punktu p
θ	-	kąt nachylenia wektora wodzącego punktu p do osi OX

Istotną komplikację z punktu widzenia obliczeń w algorytmach może stanowić wyznaczenie wartości kąta θ (ze względu na potrzebę wykorzystania funkcji trygonometrycznych), którego dokładną wartość można wyliczyć za pomocą wzoru I.1.

$$\theta(p) = \begin{cases} \operatorname{arctg} \frac{y(p)}{x(p)} & \text{dla} \quad x(p) > 0; \\ \operatorname{arctg} \frac{y(p)}{x(p)} + \pi & \text{dla} \quad x(p) < 0 \quad \wedge \quad y(p) \geqslant 0; \\ \operatorname{arctg} \frac{y(p)}{x(p)} - \pi & \text{dla} \quad x(p) < 0 \quad \wedge \quad y(p) < 0; \\ \frac{\pi}{2} & \text{dla} \quad x(p) = 0 \quad \wedge \quad y(p) < 0; \\ -\frac{\pi}{2} & \text{dla} \quad x(p) = 0 \quad \wedge \quad y(p) > 0. \end{cases} \tag{I.1}$$

Należy jednak zauważyć, że do celów sortowania wystarczy zastosować funkcję $\alpha(p)$, taką, że dla dowolnej pary punktów p_1, p_2 spełnione będą warunki I.2 oraz I.3.

$$\alpha(p_1) < \alpha(p_2) \Leftrightarrow \theta(p_1) < \theta(p_2) \tag{I.2}$$

$$\alpha(p_1) = \alpha(p_2) \Leftrightarrow \theta(p_1) = \theta(p_2) \tag{I.3}$$

Przykładowa funkcja $\alpha(p)$ zachowująca spełniająca warunki I.2 i I.3 została przedstawiona za pomocą wzoru I.4.

$$\alpha(p) = \begin{cases} \frac{y(p)}{d(p)} & \text{dla} \quad x(p) \geqslant 0 \quad \wedge \quad y(p) \geqslant 0; \\ 2 - \frac{y(p)}{d(p)} & \text{dla} \quad x(p) < 0 \quad \wedge \quad y(p) \geqslant 0; \\ 2 + \frac{|y(p)|}{d(p)} & \text{dla} \quad x(p) < 0 \quad \wedge \quad y(p) < 0; \\ 4 - \frac{|y(p)|}{d(p)} & \text{dla} \quad x(p) \geqslant 0 \quad \wedge \quad y(p) < 0. \end{cases} \tag{I.4}$$

Gdzie funkcja $d(p)$ określona jest zgodnie ze wzorem I.5.

$$d(p) = |x(p)| + |y(p)| \tag{I.5}$$

W celu udowodnienia, że funkcja $\alpha(p)$ spełnia warunki I.2 i I.3, musimy wykazać, że funkcja $\gamma(\theta)$ (przedstawiona we wzorze I.6), która stanowi przekształcenie funkcji $\alpha(p)$ takie, że jest w pełni zależna od wartości θ, a wartości $x(p(\theta))$ oraz $y(p(\theta))$ są określone zgodnie ze wzorem I.7, jest rosnąca w każdym swoim przedziale.

$$\gamma(\theta) = \alpha(p(\theta)) \tag{I.6}$$

$$p(\theta) = (|R| \cdot \cos\theta, |R| \cdot \sin\theta) \tag{I.7}$$

Należy zauważyć, że poszczególne przypadki opisane we wzorze I.4 dzielą układ współrzędnych na cztery ćwiartki. Zatem z rysunku I.2 wynikają zależności I.8, I.9, I.10 oraz I.11.

$$x(p) \geqslant 0 \ \wedge \ y(p) \geqslant 0 \ \Leftrightarrow \ \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle \tag{I.8}$$

$$x(p) < 0 \ \wedge \ y(p) \geqslant 0 \ \Leftrightarrow \ \theta \in \left(\frac{\pi}{2}; \pi \right\rangle \tag{I.9}$$

$$x(p) < 0 \ \wedge \ y(p) < 0 \ \Leftrightarrow \ \theta \in \left(\pi; \frac{3\pi}{2} \right) \tag{I.10}$$

$$x(p) \geqslant 0 \ \wedge \ y(p) < 0 \ \Leftrightarrow \ \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right) \tag{I.11}$$

Zatem w celu udowodnienia spełnienia warunków I.2 i I.3 przez funkcję $\alpha(p)$ (I.4) należy udowodnić, że funkcja $\gamma(\theta)$ (I.6) jest rosnąca (jej pochodna jest większa od zera) we wszystkich przedziałach opisanych w zależnościach I.8, I.9, I.10 oraz I.11.

Obliczenia dla tych czterech przedziałów przedstawiają się w następujący sposób. Dla każdego z przedziałów możliwe jest uproszczenie funkcji $d(p)$, tak aby pozbyć się modułu, przez co różniczkowanie funkcji $\gamma(\theta)$ staje się łatwiejsze.

1° $x(p) \geqslant 0 \ \wedge \ y(p) \geqslant 0 \ \Leftrightarrow \ \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$

$$\alpha(p) = \frac{y(p)}{d(p)} = \frac{y(p)}{|x(p)| + |y(p)|} = \frac{y(p)}{x(p) + y(p)}$$

$$\gamma(\theta) = \alpha(p(\theta)) = \frac{|R| \cdot \sin\theta}{|R| \cdot \cos\theta + |R| \cdot \sin\theta} = \frac{\sin\theta}{\sin\theta + \cos\theta}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &= \frac{d}{d\theta}\left(\frac{\sin\theta}{\sin\theta + \cos\theta}\right) \\
&= \frac{\frac{d}{d\theta}(\sin\theta) \cdot (\sin\theta + \cos\theta) - \sin\theta \cdot \frac{d}{d\theta}(\sin\theta + \cos\theta)}{(\sin\theta + \cos\theta)^2} \\
&= \frac{\cos\theta \cdot (\sin\theta + \cos\theta) - \sin\theta \cdot (\cos\theta - \sin\theta)}{\sin^2\theta + 2\sin\theta\cos\theta + \cos^2\theta} \\
&= \frac{\cos\theta\sin\theta + \cos^2\theta - \sin\theta\cos\theta + \sin^2\theta}{1 + 2\sin\theta\cos\theta} = \frac{1}{1 + 2\sin\theta\cos\theta} \\
&= \frac{1}{1 + \sin 2\theta}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &> 0 \\
\frac{1}{1 + \sin 2\theta} &> 0 \\
1 + \sin 2\theta &> 0 \\
\sin 2\theta &> -1 \\
\sin 2\theta &\neq -1 \\
2\theta &\neq \frac{3}{2}\pi + 2k\pi \\
\theta &\neq \frac{3}{4}\pi + k\pi; \ k \in \mathbb{Z}
\end{aligned}$$

$$\left.\begin{array}{l} k = 0 \ \Rightarrow \ \theta \neq \frac{3}{4}\pi > \frac{\pi}{2} \\ k = -1 \ \Rightarrow \ \theta \neq -\frac{1}{4}\pi < 0 \end{array}\right\} \Rightarrow \frac{d}{d\theta}\gamma(\theta) > 0 \text{ dla } \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$$

$$\left.\begin{array}{lcl} \gamma(0) & = & \frac{\sin 0}{\sin 0 + \cos 0} = \frac{0}{0+1} = 0 \\ \gamma\left(\frac{\pi}{2}\right) & = & \frac{\sin\frac{\pi}{2}}{\sin\frac{\pi}{2} + \cos\frac{\pi}{2}} = \frac{1}{1+0} = 1 \end{array}\right\} \Rightarrow \gamma(\theta) \in \langle 0; 1 \rangle \ \text{ dla } \theta \in \left\langle 0; \frac{\pi}{2} \right\rangle$$

$2^\circ\ \ x(p) < 0 \ \wedge\ y(p) \geqslant 0 \ \Leftrightarrow\ \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$

$$\begin{aligned}
\alpha(p) &= 2 - \frac{y(p)}{d(p)} = 2 - \frac{y(p)}{|x(p)| + |y(p)|} = 2 - \frac{y(p)}{-x(p) + y(p)} \\
&= 2 - \frac{y(p)}{y(p) - x(p)} \\
\gamma(\theta) &= \alpha(p(\theta)) = 2 - \frac{|R| \cdot \sin\theta}{|R| \cdot \sin\theta - |R| \cdot \cos\theta} = 2 - \frac{\sin\theta}{\sin\theta - \cos\theta} \\
\frac{d}{d\theta}\gamma(\theta) &= \frac{d}{d\theta}\left(2 - \frac{\sin\theta}{\sin\theta - \cos\theta}\right) \\
&= -\frac{\frac{d}{d\theta}(\sin\theta) \cdot (\sin\theta - \cos\theta) - \sin\theta \cdot \frac{d}{d\theta}(\sin\theta - \cos\theta)}{(\sin\theta - \cos\theta)^2} \\
&= -\frac{\cos\theta \cdot (\sin\theta - \cos\theta) - \sin\theta \cdot (\cos\theta + \sin\theta)}{\sin^2\theta - 2\sin\theta\cos\theta + \cos^2\theta} \\
&= -\frac{\cos\theta\sin\theta - \cos^2\theta - \sin\theta\cos\theta - \sin^2\theta}{1 - 2\sin\theta\cos\theta} \\
&= -\frac{-\cos^2\theta - \sin^2\theta}{1 - 2\sin\theta\cos\theta} = \frac{\sin^2\theta + \cos^2\theta}{1 - \sin 2\theta} = \frac{1}{1 - \sin 2\theta}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &> 0 \\
\frac{1}{1 - \sin 2\theta} &> 0 \\
1 - \sin 2\theta &> 0 \\
-\sin 2\theta &> -1 \\
\sin 2\theta &< 1 \\
\sin 2\theta &\neq 1 \\
2\theta &\neq \frac{\pi}{2} + 2k\pi \\
\theta &\neq \frac{\pi}{4} + k\pi;\ \ k \in \mathbb{Z}
\end{aligned}$$

$$\left.\begin{aligned}
k = 0 \ &\Rightarrow\ \theta \neq \tfrac{\pi}{4} < \tfrac{\pi}{2} \\
k = 1 \ &\Rightarrow\ \theta \neq \tfrac{5}{4}\pi > \pi
\end{aligned}\right\} \Rightarrow \frac{d}{d\theta}\gamma(\theta) > 0 \text{ dla } \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$$

$$\left.\begin{aligned}
\gamma\left(\tfrac{\pi}{2}\right) &= 2 - \tfrac{\sin\frac{\pi}{2}}{\sin\frac{\pi}{2} - \cos\frac{\pi}{2}} = 2 - \tfrac{1}{1-0} = 1 \\
\gamma(\pi) &= 2 - \tfrac{\sin\pi}{\sin\pi - \cos\pi} = 2 - \tfrac{0}{0-(-1)} = 2
\end{aligned}\right\} \Rightarrow \gamma(\theta) \in (1; 2\rangle \text{ dla } \theta \in \left(\frac{\pi}{2}; \pi\right\rangle$$

$3^\circ\ \ x(p) < 0 \ \wedge\ y(p) < 0 \ \Leftrightarrow\ \theta \in \left(\pi; \frac{3\pi}{2}\right)$

$$\begin{aligned}
\alpha(p) &= 2 + \frac{|y(p)|}{d(p)} = 2 + \frac{-y(p)}{|x(p)| + |y(p)|} = 2 - \frac{y(p)}{-x(p) - y(p)} \\
&= 2 + \frac{y(p)}{x(p) + y(p)} \\
\gamma(\theta) &= \alpha(p(\theta)) = 2 + \frac{|R| \cdot \sin\theta}{|R| \cdot \cos\theta + |R| \cdot \sin\theta} = 2 + \frac{\sin\theta}{\sin\theta + \cos\theta} \\
\frac{d}{d\theta}\gamma(\theta) &= \frac{d}{d\theta}\left(2 + \frac{\sin\theta}{\sin\theta + \cos\theta}\right) \\
&= \frac{\frac{d}{d\theta}(\sin\theta) \cdot (\sin\theta + \cos\theta) - \sin\theta \cdot \frac{d}{d\theta}(\sin\theta + \cos\theta)}{(\sin\theta + \cos\theta)^2} \\
&= \frac{\cos\theta \cdot (\sin\theta + \cos\theta) - \sin\theta \cdot (\cos\theta - \sin\theta)}{\sin^2\theta + 2\sin\theta\cos\theta + \cos^2\theta} \\
&= \frac{\cos\theta\sin\theta + \cos^2\theta - \sin\theta\cos\theta + \sin^2\theta}{1 + 2\sin\theta\cos\theta} \\
&= \frac{\cos^2\theta + \sin^2\theta}{1 + 2\sin\theta\cos\theta} = \frac{1}{1 + \sin 2\theta}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &> 0 \\
\frac{1}{1 + \sin 2\theta} &> 0 \\
1 + \sin 2\theta &> 0 \\
\sin 2\theta &> -1 \\
\sin 2\theta &\neq -1 \\
2\theta &\neq \frac{3}{2}\pi + 2k\pi \\
\theta &\neq \frac{3}{4}\pi + k\pi;\ k \in \mathbb{Z}
\end{aligned}$$

$$\left.\begin{array}{lcl}
k = 0 & \Rightarrow & \theta \neq \frac{3}{4}\pi < \pi \\
k = 1 & \Rightarrow & \theta \neq \frac{7}{4}\pi > \frac{3\pi}{2}
\end{array}\right\} \Rightarrow \frac{d}{d\theta}\gamma(\theta) > 0 \text{ dla } \theta \in \left(\pi; \frac{3\pi}{2}\right)$$

$$\left.\begin{array}{rcl}
\gamma(\pi) & = & 2 + \frac{\sin\pi}{\sin\pi + \cos\pi} = 2 - \frac{0}{0-1} = 2 \\
\gamma\left(\frac{3\pi}{2}\right) & = & 2 + \frac{\sin\frac{3\pi}{2}}{\sin\frac{3\pi}{2} + \cos\frac{3\pi}{2}} \\
 & = & 2 + \frac{-1}{-1+0} = 2 + 1 = 3
\end{array}\right\} \Rightarrow \gamma(\theta) \in (2; 3) \ \text{ dla } \theta \in \left(\pi; \frac{3\pi}{2}\right)$$

4° $x(p) \geqslant 0 \;\wedge\; y(p) < 0 \;\Leftrightarrow\; \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right)$

$$\alpha(p) = 4 - \frac{|y(p)|}{d(p)} = 4 - \frac{-y(p)}{|x(p)| + |y(p)|} = 4 + \frac{y(p)}{x(p) - y(p)}$$

$$\gamma(\theta) = \alpha(p(\theta)) = 4 + \frac{|R| \cdot \sin\theta}{|R| \cdot \cos\theta - |R| \cdot \sin\theta} = 4 + \frac{\sin\theta}{\cos\theta - \sin\theta}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &= \frac{d}{d\theta}\left(4 + \frac{\sin\theta}{\cos\theta - \sin\theta}\right) \\
&= \frac{\frac{d}{d\theta}(\sin\theta) \cdot (\cos\theta - \sin\theta) - \sin\theta \cdot \frac{d}{d\theta}(\cos\theta - \sin\theta)}{(\cos\theta - \sin\theta)^2} \\
&= \frac{\cos\theta \cdot (\cos\theta - \sin\theta) - \sin\theta \cdot (-\sin\theta - \cos\theta)}{\cos^2\theta - 2\cos\theta\cos\theta + \sin^2\theta} \\
&= \frac{\cos^2\theta - \cos\theta\sin\theta + \sin^2\theta + \sin\theta\cos\theta}{1 - 2\sin\theta\cos\theta} \\
&= \frac{\cos^2\theta + \sin^2\theta}{1 - 2\sin\theta\cos\theta} = \frac{1}{1 - \sin 2\theta}
\end{aligned}$$

$$\begin{aligned}
\frac{d}{d\theta}\gamma(\theta) &> 0 \\
\frac{1}{1 - \sin 2\theta} &> 0 \\
1 - \sin 2\theta &> 0 \\
-\sin 2\theta &> -1 \\
\sin 2\theta &< 1 \\
\sin 2\theta &\neq 1 \\
2\theta &\neq \frac{\pi}{2} + 2k\pi \\
\theta &\neq \frac{\pi}{4} + k\pi;\ k \in \mathbb{Z}
\end{aligned}$$

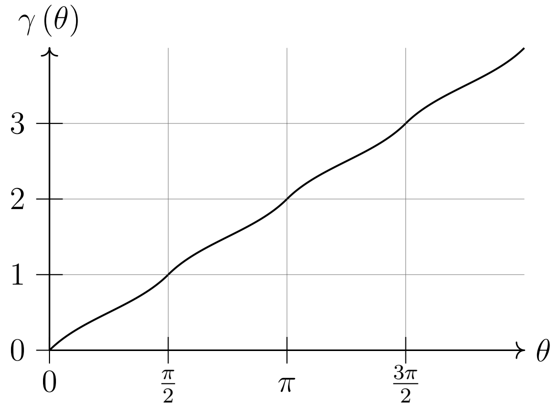
$$\left.\begin{array}{lcl} k = 1 & \Rightarrow & \theta \neq \frac{5}{4}\pi > \frac{3\pi}{2} \\ k = 2 & \Rightarrow & \theta \neq \frac{9}{4}\pi > 2\pi \end{array}\right\} \Rightarrow \frac{d}{d\theta}\gamma(\theta) > 0 \text{ dla } \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right)$$

$$\left.\begin{array}{rcl} \gamma\left(\frac{3\pi}{2}\right) & = & 4 + \frac{\sin\frac{3\pi}{2}}{\cos\frac{3\pi}{2} - \sin\frac{3\pi}{2}} \\ & = & 4 + \frac{-1}{0-(-1)} = 4 - 1 = 3 \\ \gamma(2\pi) & = & 4 + \frac{\sin 2\pi}{\cos 2\pi - \sin 2\pi} \\ & = & 4 + \frac{0}{1-0} = 4 \end{array}\right\} \Rightarrow \gamma(\theta) \in \langle 3; 4) \text{ dla } \theta \in \left\langle \frac{3\pi}{2}; 2\pi \right)$$

Z powyższych obliczeń wynika, że we wszystkich przedziałach z zależności I.8, I.9, I.10 oraz I.11 funkcja $\gamma\left(\theta\right)$ ma dodatnią pochodną, a tym samym jest rosnąca we wszystkich przedziałach, oraz każda z wartości osiąganych w przedziale jest większa od każdej wartości z poprzedniego przedziału, a tym samym funkcja $\alpha\left(p\right)$ spełnia warunki I.2 i I.3.

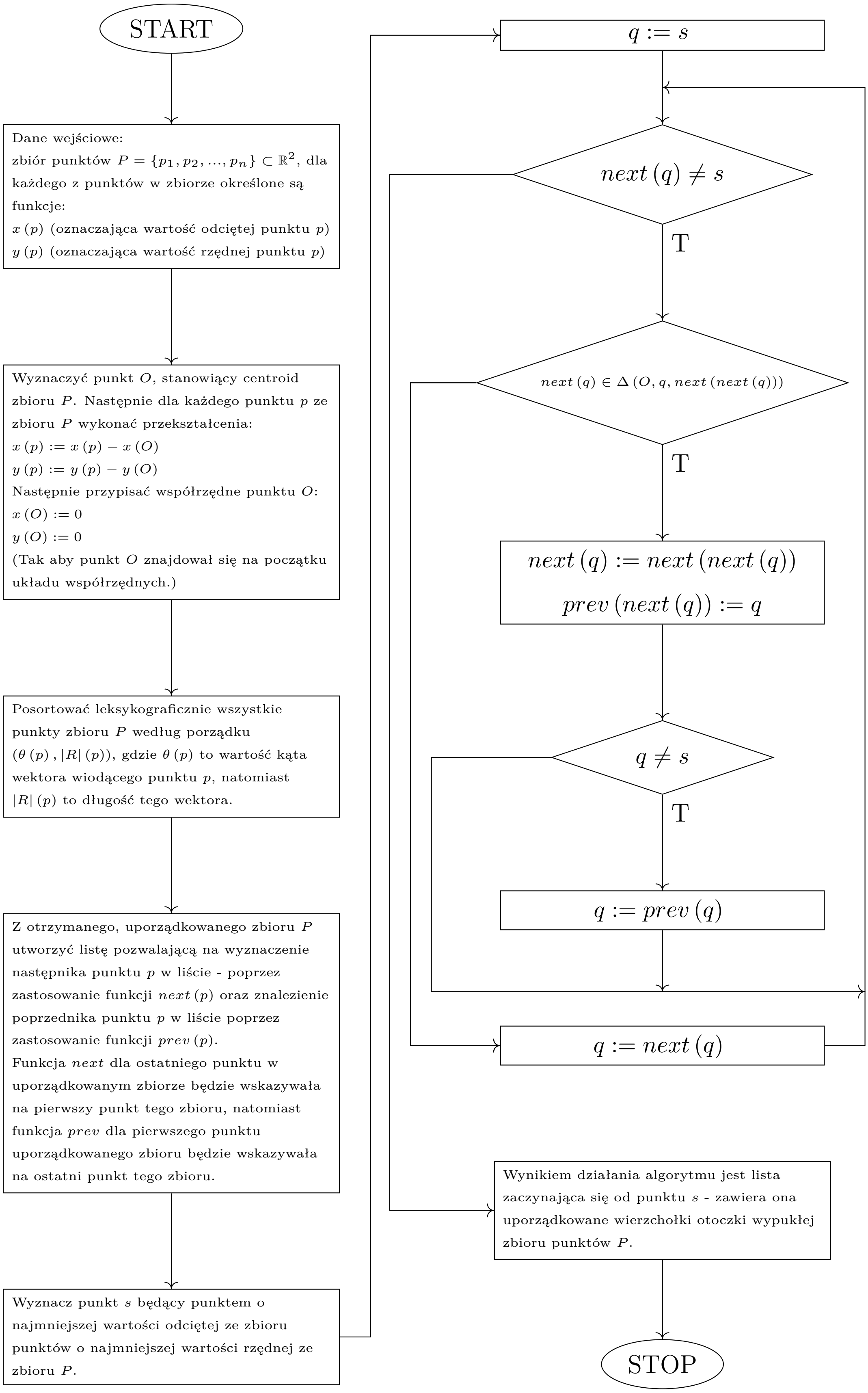
Postać funkcji $\gamma\left(\theta\right)$ może być opisana wzorem I.12 a wykres jej przebiegu, przedstawiony na rysunku I.3 wyraźnie potwierdza jej rosnącą monotoniczność we wszystkich przedziałach.

$$\gamma\left(\theta\right)=\begin{cases}\frac{\sin\theta}{\sin\theta+\cos\theta} & \text{dla} \quad \theta\in\left\langle 0;\frac{\pi}{2}\right\rangle \\ 2-\frac{\sin\theta}{\sin\theta-\cos\theta} & \text{dla} \quad \theta\in\left(\frac{\pi}{2};\pi\right\rangle \\ 2+\frac{\sin\theta}{\sin\theta+\cos\theta} & \text{dla} \quad \theta\in\left(\pi;\frac{3\pi}{2}\right) \\ 4-\frac{\sin\theta}{\sin\theta+\cos\theta} & \text{dla} \quad \theta\in\left\langle\frac{3\pi}{2};2\pi\right)\end{cases} \tag{I.12}$$



Rysunek I.3: Wykres funkcji $\gamma\left(\theta\right)$
Źródło: opracowanie własne

1.1 Algorytm Grahama



Rysunek I.4: Algorytm Grahama

   1.2 Algorytm Jarvisa

2. Otoczka wypukła wielokąta prostego

3. Redukcja zbioru punktów do wielokąta prostego

Rozdział II
Zastosowania

Istotnym, jeśli nie najistotniejszym zagadnieniem dotyczącym otoczek wypukłych są ich zastosowania. Już od kilkudziesięciu lat problem ten temat znajduje swoje użycie w wielu dziedzinach kombinatoryki i informatyki. Dla przykładu problem sortowania elementów liczbowych listy można sprowadzić do problemu znalezienia otoczki wypukłej. W związku z tym rozwiązanie tego problemu może pomóc w rozwiązaniu innych problemów w bardziej symboliczny i graficzny sposób.

1. Generalizacja kartograficzna

W celu jak najbardziej informatywnego i niezłożonego przedstawienia danych geograficznych w sposób graficzny potrzebne jest zastosowanie algorytmu generalizującego informacje.

2. Grafika komputerowa

Obiekty wykorzystywane w grafice komputerowej często mogą charakteryzować się skomplikowanymi kształtami. Im bardziej skomplikowany kształt, tym więcej mocy obliczeniowej potrzeba w celu wykonania danej operacji na tym kształcie. W niektórych przypadkach do uproszczenia graficznego danego obiektu używa się otoczki wypukłej jego kształtu. Ze względu na fakt, iż otoczka wypukła wielokąta zawsze będzie miała liczbę wierzchołków mniejszą lub równą liczbie wierzchołków samego wielokąta, powstała otoczka może posłużyć do wykonania mniejszej liczby obliczeń przy wykonywaniu operacji na danym obiekcie.

3. Detekcja obiektów

4. Wyznaczanie obwiedni sygnału

Sygnał to zapis informacji zmieniającej się w zależności od czasu. Taką informacją może być natężenie/napięcie prądu, natężenie pola elektromagnetycznego, dźwięk utworu muzycznego itd. W przypadku, jeśli zmienną informację da się reprezentować w sposób liczbowy, możliwe jest graficzne przedstawienie sygnału. Zapisanie informacji w ten sposób może być łatwiejsze do interpretacji przez człowieka. Jednak w przypadku, gdy mamy do czynienia ze skomplikowanym przebiegiem, na

Algorytm wyznaczający otoczkę wypukłą wielokąta prostego może posłużyć znajdowaniu obwiedni sygnału.

Rozdział III
Implementacja w języku Scala

1. Pojęcia ogólne

W tej sekcji przedstawione zostaną listingi zawierające kod w języku Scala definiujący pojęcia potrzebne do implementacji poszczególnych algorytmów wyznaczających otoczkę wypukłą.

Klasa `Point` posłuży jako podstawowa klasa reprezentująca punkt w przestrzeni dwuwymiarowej. W celu łatwiejszego manipulowania danymi. Zostały w jej ramach zaimplementowane 3 metody pomocnicze.

Odejmowanie (linia 7) - zaimplementowane jako odejmowanie od siebie odpowiadających sobie współrzędnych dwu punktów.

Dodawanie (linia 8) - dodawanie do siebie odpowiadających sobie współrzędnych dwu punktów.

Dzielenie przez skalar (linia 9) - zaimplementowane jako dzielenie obu współrzędnych przez wskazaną liczbę. Zaimplementowane dzięki użyciu domniemanemu argumentowi `numeric: Numeric[T]`. W języku Scala tego typu konstrukcja umożliwia korzystanie z metody `/` w tedy i tylko wtedy jeśli zaimportowany zostanie do aktualnego kontekstu obiekt typu `Numeric[T]`. Domyślnie typ ten jest zaimplementowany dla `T` będącym jednym z podstawowych typów liczbowych: `Float`, `Int`, `Double` etc. Więc w domyśle tego typu konstrukcja służy niejako „udowodnieniu”, że wskazywany argument jest w istocie liczbą.

```
1  package utilities.geometry
2  
3  case class Point(
4                    x: Double,
5                    y: Double
6                  ) {
7    def -(that: Point): Point = Point(x - that.x, y - that.y)
8    def +(that: Point): Point = Point(x + that.x, y + that.y)
9    def /[T](number: T)(implicit numeric: Numeric[T]): Point = {
10     val double = numeric.toDouble(number)
```

```
    Point(x / double, y / double)
  }
}
```

Kolejnym istotnym elementem wykorzystanym w implementacji algorytmów wyznaczających otoczkę wypukłą zbioru punktów na płaszczyźnie są metody wyliczające określone właściwości punktów.

Metoda `calculateCentroid(points: Points): Points` (linia 11) wyznacza centroid zbioru punktów `points`.

Metoda `distanceFromCenterSquared(p: Point): Double` wyznacza kwadrat odległości wskazanego punktu od środka układu współrzędnych.

Na podstawie wyżej wymienionej metody zostały zaimplementowane dwie metody związane z odległością między punktami.

Metoda

```
1  package utilities.geometry
2
3  import java.lang.Math._
4
5  object PointsUtils {
6
7    type Points = List[Point]
8
9    private lazy val HALF_PI = PI * .5
10
11   def calculateCentroid(points: Points): Point = points.reduceLeft(
       _ + _) / points.length
12
13   def distanceSquared(a: Point, b: Point): Double =
       distanceFromCenterSquared(a - b)
14
15   def distance(a: Point, b: Point): Double = distanceFromCenter(a -
       b)
16
17   def distanceFromCenterSquared(p: Point): Double = pow(p.x, 2) +
       pow(p.y, 2)
18
19   def distanceFromCenter(p: Point): Double = sqrt(
       distanceFromCenterSquared(p))
20
21   def phase(p: Point): Double = {
22     import p.{x, y}
```

```
    (x.sign, y.sign) match {
      case (1, _) => atan(y / x)
      case (-1, 0 | 1) => atan(y / x) + PI
      case (-1, -1) => atan(y / x) - PI
      case (0, -1) => HALF_PI
      case (0, 1) => -HALF_PI
    }
  }

  def alpha(p: Point): Double = {
    import p.{x, y}
    val absX = abs(x)
    val absY = abs(y)
    val d = absX + absY

    (x.sign, y.sign) match {
      case (0 | 1, 0 | 1) => y / d
      case (-1, 0 | 1) => 2 - y / d
      case (-1, -1) => 2 + absY / d
      case (0 | 1, -1) => 4 - absY / d
    }
  }
}
```

```
package utilities.geometry.ordering

import utilities.geometry.{Point, PointsUtils}
import Ordering.Double.TotalOrdering

trait OrientationOrdering extends Ordering[Point] {
  final override def compare(x: Point, y: Point): Int = {
    TotalOrdering.compare(phase(x), phase(y)) match {
      case 0 => TotalOrdering.compare(distance(x), distance(y))
      case phaseCompared => phaseCompared
    }
  }

  protected def distance(p: Point): Double
  protected def phase(p: Point): Double
}

object OrientationOrdering {
  implicit object Exact extends OrientationOrdering {
    override protected def distance(p: Point): Double = PointsUtils.distanceFromCenter(p)
    override protected def phase(p: Point): Double = PointsUtils.phase(p)
  }

  implicit object Indicator extends OrientationOrdering {
    override protected def distance(p: Point): Double = PointsUtils.distanceFromCenterSquared(p)
    override protected def phase(p: Point): Double = PointsUtils.alpha(p)
  }
}
```

```
package utilities.geometry.convexhull.algorithms

import utilities.geometry.PointsUtils.Points
import utilities.geometry.ordering.OrientationOrdering

trait ConvexHullAlgorithm extends Product {
  def calculate(points: Points)(implicit ordering: OrientationOrdering): Points
}
```

```scala
package utilities.geometry.convex.hull.algorithms

import org.scalatest.Assertion
import org.scalatest.wordspec.AnyWordSpec
import utilities.geometry.Point
import utilities.geometry.PointsUtils.Points
import AlgorithmTest.{EXPECTED_CONVEX_HULL, INPUT_POINTS, ONE_INPUT_POINT}
import utilities.geometry.convexhull.algorithms.ConvexHullAlgorithm
import utilities.geometry.ordering.OrientationOrdering

abstract class AlgorithmTest(convexHullAlgorithm: ConvexHullAlgorithm)(implicit ordering: OrientationOrdering) extends AnyWordSpec {
  private def convexHullAlgorithmName: String = convexHullAlgorithm.productPrefix

  private def assertExpectedOutput(input: Points, expectedOutput: Points): Assertion = {
    val actualConvexHull = convexHullAlgorithm.calculate(input)
    assert(actualConvexHull == expectedOutput)
  }

  s"$convexHullAlgorithmName" must {
    "throw the IllegalArgumentException for empty" in {
      assertThrows[IllegalArgumentException](convexHullAlgorithm.calculate(Nil))
    }

    "return the same thing for one input point" in {
      assertExpectedOutput(ONE_INPUT_POINT, ONE_INPUT_POINT)
    }

    "properly calculate convex hull" in {
      assertExpectedOutput(INPUT_POINTS, EXPECTED_CONVEX_HULL)
    }
  }
}
```

```

object AlgorithmTest {
  val INPUT_POINTS: Points = List(
    Point(1, 3),
    Point(2, 4),
    Point(2, 2),
    Point(4, 6),
    Point(3, 4),
    Point(4, 4),
    Point(4, 3),
    Point(2, 6),
    Point(5, 2),
    Point(6, 4),
    Point(5, 5)
  )

  val EXPECTED_CONVEX_HULL: Points = List(
    Point(6, 4),
    Point(4, 6),
    Point(2, 6),
    Point(1, 3),
    Point(2, 2),
    Point(5, 2)
  )

  val ONE_INPUT_POINT: Points = List(
    Point(1, 2)
  )
}
```

2. Algorytm Grahama

```
package utilities.geometry.convexhull.algorithms
import utilities.geometry.PointsUtils.Points
import utilities.geometry.ordering.OrientationOrdering
import utilities.geometry.{PointsCycle, PointsUtils}


case object Graham extends ConvexHullAlgorithm {

  override def calculate(points: Points)(implicit ordering:
    OrientationOrdering): Points = {
    require(points.nonEmpty, "You can not calculate convex hull of
    an empty points set.")
    val centroid = PointsUtils.calculateCentroid(points)
    val pointsSorted = points.map(_ - centroid).sorted
    val pointsCycle = new PointsCycle(pointsSorted)

    pointsCycle.getHull.map(_ + centroid)
  }
}
```

```
package utilities.geometry

import utilities.geometry.PointsUtils.Points

class PointsCycle(points: Points) extends Iterable[Point] {

  private lazy val first = NoPointRef.next

  override def iterator: Iterator[Point] = new Iterator[Point] {
    private var currentPointRef: AbstractPointRef = NoPointRef

    override def hasNext: Boolean = currentPointRef.isNotLast

    override def next(): Point = {
      val next = currentPointRef.next
      currentPointRef = next
      next.point
    }
  }

  private def createPointRefs(points: Points): AbstractPointRef =
    points.foldLeft[AbstractPointRef](NoPointRef)(_.addPoint(_)).addLastPointAndReturn()

  createPointRefs(points)

  trait AbstractPointRef {
    var next: PointRef = _
    var prev: PointRef = _

    def isNotFirst: Boolean
    def isNotLast: Boolean

    def addPoint(point: Point): PointRef

    protected def addLastPoint(): Unit

    final def addLastPointAndReturn(): AbstractPointRef = {
      addLastPoint()
```

```
      this
    }
  }

  class PointRef(val point: Point) extends AbstractPointRef {

    final def isNotFirst: Boolean = this ne first
    final def isNotLast: Boolean = next ne first

    override def addPoint(point: Point): PointRef = {
      val pointRef = new PointRef(point)
      pointRef.prev = this
      next = pointRef
      pointRef
    }

    def isInTriangle(t2: PointRef, t3: PointRef): Boolean =
      Orientation.calculateThreePointsOrientation(t2.point, t3.
   point, point) == Orientation.Left

    override protected def addLastPoint(): Unit = {
      next = first
      first.prev = this
    }
  }

  object NoPointRef extends AbstractPointRef {

    override def addPoint(point: Point): PointRef = {
      val pointRef = new PointRef(point)
      next = pointRef
      pointRef
    }

    override def isNotFirst: Boolean = first ne null
    override def isNotLast: Boolean = first ne null

    override protected def addLastPoint(): Unit = ()
  }
```

```

  def getHull: Points = {

    if (NoPointRef.isNotLast) {
      var q = NoPointRef.next
      while (q.isNotLast) {
        if (q.next.isInTriangle(q, q.next.next)) {
          q.next = q.next.next
          q.next.prev = q
          if (q.isNotFirst) q = q.prev
        } else q = q.next
      }
    }
    toList
  }

}
```

```
package utilities.geometry.convex.hull.algorithms

import utilities.geometry.convexhull.algorithms.Graham
import utilities.geometry.ordering.OrientationOrdering.Indicator

class GrahamAlgorithmTest extends AlgorithmTest(Graham)
```

3. Algorytm Jarvisa

Dzięki wykorzystaniu pattern matchingu dostępnego w Scali, możliwe jest przejrzyste zaimplementowanie algorytmu Jarvisa, który, jak widać jest dużo mniej skomplikowany od algorytmu Grahama. Mniejsze skomplikowanie implementacji wiąże się jednak z dużo większą złożonością obliczeniową.

```
package utilities.geometry.convexhull.algorithms

import utilities.geometry.{Point, PointsUtils}
import utilities.geometry.PointsUtils.Points

case class Jarvis() extends ConvexHullAlgorithm {

  private val FIRST_POINT_ORDERING = new Ordering[Point] {
    override def compare(x: Point, y: Point): Int = {
      val byX = Ordering.Double.TotalOrdering.compare(x.x, y.x)
      lazy val byY = Ordering.Double.TotalOrdering.compare(x.y, y.y)
      if (byX == 0) byY
      else byX
    }
  }


  override protected def nonEmptyCalculate(points: Points): Points = {
    val firstPoint = points.max(FIRST_POINT_ORDERING)
    val pZero = Point(firstPoint.x - 1, firstPoint.y)

    var pI = firstPoint
    var pIMinusOne = pZero
```

```

    val pointsBuilder = List.newBuilder[Point]
    pointsBuilder += firstPoint

    while (true) {
      points
        .filterNot(_ == pI)
        .maxByOption(PointsUtils.angleBetweenThreePoints(pIMinusOne
    , pI, _)) match {
        case Some(`firstPoint`) => return pointsBuilder.result()
        case Some(pIPlusOne) =>
          pointsBuilder += pIPlusOne
          pIMinusOne = pI
          pI = pIPlusOne
        case _ => return pointsBuilder.result()
      }
    }
    Nil
  }

}
```

```
package utilities.geometry.convex.hull.algorithms

import utilities.geometry.convexhull.algorithms.Jarvis

class JarvisAlgorithmTest extends AlgorithmTest(Jarvis())
```

Rozdział IV
Dynamiczna otoczka wypukła

1. Algorytm

2. Implementacja w języku Scala

Rozdział V
Podsumowanie

Bibliografia

[1] Ronald L. Graham, Frances Yao, Finding the Convex Hull of a Simple Polygon (1981)

[2] Avraham A. Melkman, On-line Construction of the Convex Hull of a Simple Polyline (1985)

[3] Jacqueleen Jourban, Yair Gabay, A Method for Construction of 2D Hull For Generalized Cartographic Representation (2000)

[4] Min Tang, Jie-yi Zhao, Ruo-feng Tong, Dinesh Manocha, GPU accelerated convex hull computation (2012)

[5] Navjot Singh, Rinki Arya, R.K. Agrawal, A convex hull approach in conjunction with Gaussian mixture model for salient object detection (2016)

[6] Fan Cheng, Qiangqiang Zhang, Ye Tian, Xingyi Zhang, Maximizing receiver operating characteristics convex hull via dynamic reference point-based multi-objective evolutionary algorithm (2019)