POLITECHNIKA WARSZAWSKA

WYDZIAŁ
MECHANICZNY ENERGETYKI I LOTNICTWA

ZAKŁAD AERODYNAMIKI

PROJEKT OBLICZENIOWY

Wyznaczenie odległości od brzegu w oparciu o równania różniczkowe cząstkowe.

Autor: inż. Marcin Stelmaszyk
Prowadzący: dr inż. Jerzy Majewski

Warszawa 2013

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis projektu

Odległość od brzegu, d, pozostaje nadal kluczowym parametrem przy modelowaniu turbulencji oraz generowaniu siatek obliczeniowych[1].

Celem niniejszego projektu jest opracowanie programu umożliwiającego wyznaczanie minimalnej odległości między danym węzłem siatki, a brzegiem profilu lotniczego.

Najprostszym sposobem rozwiązania powyższego zagadnienia jest bezpośrednie obliczenie odległości d danego węzła P od kolejnych węzłów S należących do brzegu profilu korzystając z miary euklidesowej:

$$d(P,S) = \sqrt{(x_p - x_s)^2 + (y_p - y_s)^2} \tag{1.1}$$

a następnie wybranie najmniejszej wartości z utworzonego zbioru odległości $\{d_0, d_1, ..., d_n\}$. Metoda ta, znana w terminologii algorytmicznej jako *metoda siłowa* (ang. *Brute Force*), jest łatwa do zaimplementowania, a liczba operacji potrzebnych do jej wykonania dla siatek stałych w czasie jest mała w stosunku do kosztu całego rozwiązania. Niemniej jednak, szybkie znalezienie odległości od brzegu ma kluczowe znaczenie dla problemów rozwiązywanych przy użyciu siatek deformujących się i adaptacyjnych, dla których wyznaczenie minimalnej odległości od brzegu jest przeprowadzane wielokrotnie[2].

Innym podejściem jest wykorzystanie równań różniczkowych cząstkowych. Do rozwiązania problemu Sethain[3] zaproponował wykorzystanie równania Eikonału następującej postaci

$$|\nabla\phi| = 1 + \lambda\nabla^2\phi \tag{1.2}$$

[1] **Tucker**.
[2] **Ibid**.
[3] **Sethain**.

Niniejszy projekt skupi się na implementacji uproszczonej jego wersji, a mianowicie równaniu Poissona na płaszczyźnie (x, y)

$$\nabla^2 \phi = -1 \tag{1.3}$$

Pomocniczo została utworzona siatka obliczeniowa wokół profilu, o strukturze dopasowanej do zadanego brzegu obszaru. Tak zdefiniowana przestrzeń fizyczna została przetransformowana do przestrzeni obliczeniowej, a następnie zdyskretyzowana metodą różnic skończonych. Powstały układ równań algebraicznych rozwiązano metodą iteracyjną. Otrzymane wyniki skonfrontowano z rozwiązaniem siłowym.

Program został zaimplementowany w języku C++ przy użyciu darmowego środowiska programistycznego Microsoft Visual Studio Express 2012[4]. Język C++ ma szerokie zastosowanie w programach obliczeniowych, więc jego wykorzystanie umożliwia integrację w istniejących lub przyszłych kodach obliczeniowych. Do wizualizacji wyników skorzystano z programu Tecplot 360[5].

Niniejsza dokumentacja została stworzona w języku LaTeX przy użyciu edytora TexMaker 3.5.2[6].

[4] `www.microsoft.com/visualstudio/`

[5] `www.tecplot.com`

[6] `http://www.xm1math.net/texmaker/`

Rozdział 2

Model matematyczny

2.1 Równanie Eikonału

Zmienna ϕ w równaniu Eikonału (1.2) modeluje czas przybycia frontu fali propagującej od powierzchni brzegu. Prawa strona równania wskazuje na jednostkową prędkość tego frontu, tzn. istnieje pole prędkości o $|\mathbf{U}| = 1$, co oznacza, że wyznaczona wartość jest równa szukanej odległości d[1].

Zmodyfikowana forma równania z jawnym laplasjanem, jak poniżej, jest zwana równaniem Hamiltona-Jacobiego

$$\mathbf{U} \circ \nabla\phi = 1 + \Gamma(\phi)\nabla^2\phi \tag{2.1}$$

Rozwiązanie (2.1), w oparciu o początkowy rozkład wartości ϕ, otrzymuje się w sposób iteracyjny.

2.2 Równanie Poissona

Przy założeniu, że $\mathbf{U} = 0$ oraz $\Gamma = 1$, równanie (2.1) można sprowadzić do równania Poissona

$$\nabla^2\phi = -1 \iff \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -1 \tag{2.2}$$

Wstawiając wartości otrzymanego rozwiązania ϕ do zależności

$$d = -\sqrt{\left(\frac{\partial\phi}{\partial x}\right)^2 + \left(\frac{\partial\phi}{\partial y}\right)^2} + \sqrt{\left(\frac{\partial\phi}{\partial x}\right)^2 + \left(\frac{\partial\phi}{\partial y}\right)^2 + 2\phi} \tag{2.3}$$

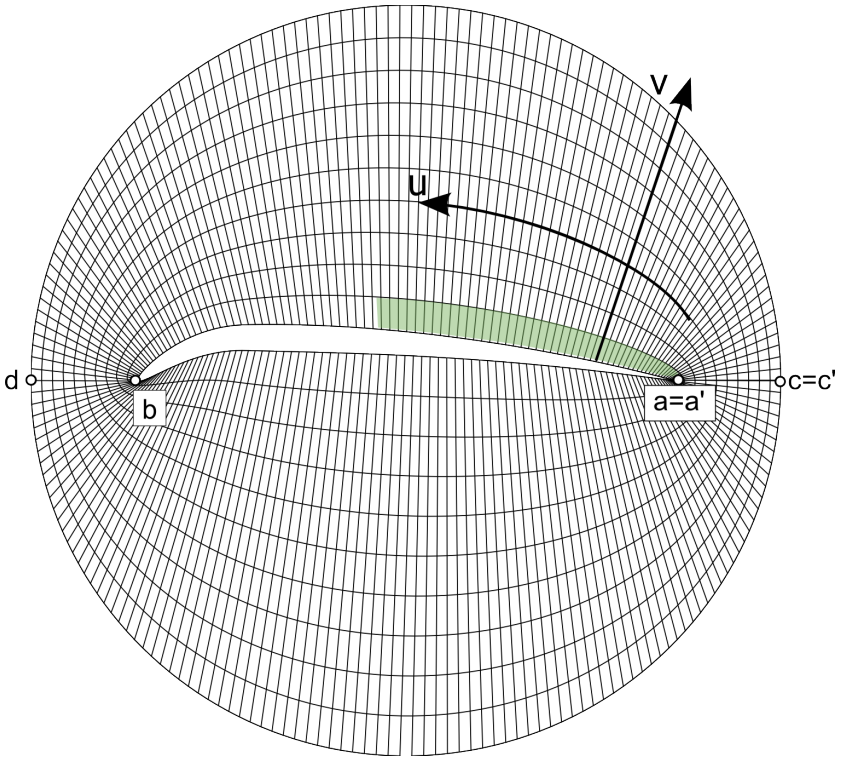
otrzymuje się poszukiwaną odległość od brzegu. Należy mieć na uwadze, że wyprowadzenie wzoru (2.3) wymaga przyjęcia nieskończonych współrzędnych poza kierunkiem normalnym do brzegu. Oznacza to, że rozwiązanie jest dokładne tylko blisko ściany. Nie stanowi to znaczącego problemu w modelach turbulencji, bowiem korzystają one tylko z wartości d wyznaczonych blisko brzegu[2]

[1] **Tucker**.

[2] A ściślej - maksymalnie do jednej trzeciej grubości warstwy przyściennej (**ibid.**)

2.3 Transformacja siatki z przestrzeni fizycznej do obliczeniowej

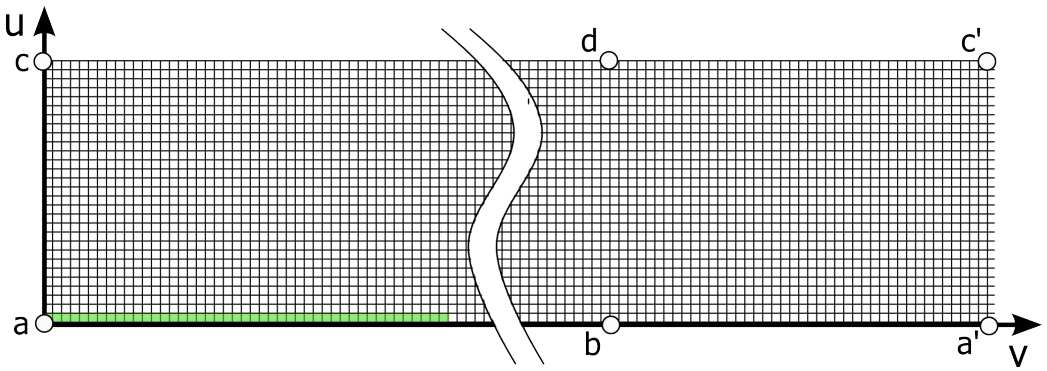
W ramach niniejszego projektu została wygenerowana strukturalna siatka obliczeniowa o topologii typu O (ang. *O-grid topology*), charakteryzująca się tym, że każdy z jej rzędów ($v = const$) tworzy wokół profilu krzywą zamkniętą, natomiast druga rodzina krzywych ($u = const$) rozciąga się prostoliniowo od powierzchni profilu (rys. 2.1). Krzywa $v = 0$ pokrywa się z obwiednią profilu (od punktu a do a'), a linie $u = 0$ (węzły a - c) oraz $u = u_{max}$ (węzły a' - c') tną siatkę w przestrzeni obliczeniowej. Wadą topologii typu O jest słaba jakość siatki w miejscu ostrej krawędzi spływu, co wpływa na poprawność rozwiązania w tym obszarze[3][4].



Rysunek 2.1: Przestrzeń fizyczna. W oparciu o węzły leżące na konturze profilu została wygenerowana siatka krzywoliniowa o topologii O. Strzałkami oznaczono kierunki zmiennych (u, v) z przestrzeni obliczeniowej.

[3] **Blazek**; **s. 359**.

[4] Należy mieć na uwadze, że dobranie powyższego typu siatki, oraz sam fakt jej generowania w programie, jest czynnością pomocniczą wobec celu niniejszego projektu i służy jedynie sprawdzeniu poprawności implementacji modelu matematycznego.



Rysunek 2.2: Przestrzeń obliczeniowa (u, v). Węzły a,a',b,c,c',d odpowiadają węzłom z rys. (2.1).

Przyjętej w projekcie metody różnic skończonych nie da się bezpośrednio wykorzystać na siatce krzywoliniowej[5]. Wymagane jest przeprowadzenie transformacji siatki niejednorodnej do postaci prostokątnej, jak i samego równania Poissona, tak aby było spełnione w nowym układzie kartezjańskim (przekształcenie z przestrzeni fizycznej do obliczeniowej)[6].

Ogólna postać operatora laplasjanu we współrzędnych krzywoliniowych (u, v) wygląda następująco[7]

$$\nabla^2 = \frac{1}{h_1 h_2}\left[\frac{\partial}{\partial u}\left(\frac{h_2}{h_1}\cdot\frac{\partial}{\partial u}\right) + \frac{\partial}{\partial v}\left(\frac{h_2}{h_1}\cdot\frac{\partial}{\partial v}\right)\right] \tag{2.4}$$

Po wykonaniu różniczkowania wyrażeń w nawiasach okrągłych otrzymujemy jego rozwiniętą formę

$$\begin{aligned}\nabla^2 = \underbrace{\frac{1}{h_1^2}}_{A}\frac{\partial^2}{\partial u^2} + \underbrace{\frac{1}{h_2^2}}_{B}\frac{\partial^2}{\partial v^2} + \underbrace{\frac{1}{h_1 h_2}\left[\frac{1}{h_1}\frac{\partial h_2}{\partial u} + h_2\frac{\partial}{\partial u}\left(\frac{1}{h_1}\right)\right]}_{C}\frac{\partial}{\partial u} \\ + \underbrace{\frac{1}{h_1 h_2}\left[\frac{1}{h_2}\frac{\partial h_1}{\partial v} + h_1\frac{\partial}{\partial v}\left(\frac{1}{h_2}\right)\right]}_{D}\frac{\partial}{\partial v}\end{aligned} \tag{2.5}$$

[5] **Anderson**; **s. 170**.
[6] **Blazek**; **s. 36**.
[7] `http://www.maths.qmul.ac.uk/~wjs/MTH5102/curvcoord10.pdf`

Występujące powyżej czynniki skalujące dane są następująco

$$h_1 = \sqrt{\left(\frac{\partial x}{\partial u}\right)^2 + \left(\frac{\partial y}{\partial u}\right)^2} \qquad h_2 = \sqrt{\left(\frac{\partial x}{\partial v}\right)^2 + \left(\frac{\partial y}{\partial v}\right)^2} \tag{2.6}$$

Pochodne czynników skalujących po współrzędnych obliczeniowych u, v można wyrazić w postaci zależnej od współrzędnych z przestrzeni fizycznej x, y

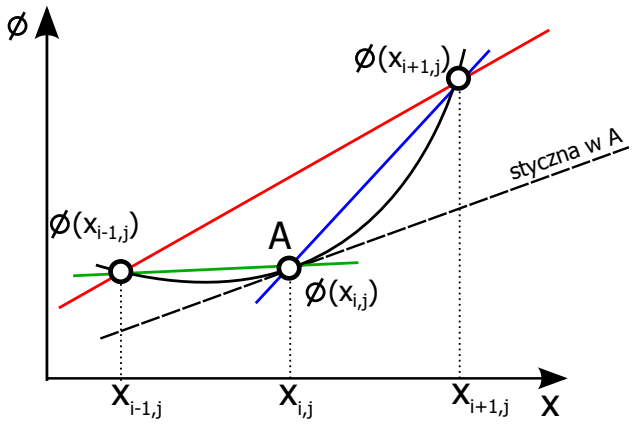
$$\begin{aligned}
\frac{\partial h_2}{\partial u} &= \frac{\partial}{\partial u}\left(\sqrt{x_v^2 + y_v^2}\right) = \frac{x_v \cdot x_{uv} + y_v \cdot y_{vv}}{h_2} \\
\frac{\partial h_1}{\partial v} &= \frac{x_u \cdot x_{uv} + y_u \cdot y_{uv}}{h_1} \\
\frac{\partial}{\partial u}\left(\frac{1}{h_1}\right) &= \frac{\partial}{\partial u}\left(\frac{1}{\sqrt{x_u^2 + y_u^2}}\right) = -\frac{x_u \cdot x_{uu} + y_u \cdot y_{uu}}{h_1^3} \\
\frac{\partial}{\partial v}\left(\frac{1}{h_2}\right) &= -\frac{x_v \cdot x_{vv} + y_v \cdot y_{vv}}{h_2^3}
\end{aligned} \tag{2.7}$$

Do obliczenia wartości poszczególnych pochodnych cząstkowych współrzędnych $x = x(u, v), y = y(u, v)$ po u, v niezbędne jest użycie metody różnic skończonych.

2.4 Dyskretyzacja metodą różnic skończonych

W metodzie różnic skończonych operatory różniczkowania zastępowane są odpowiednimi operatorami różnicowymi. W rezultacie otrzymuje się układ równań algebraicznych, który w przeciwieństwie do układu równań różniczkowych jest prostszy i w większości przypadków w ogóle możliwy do rozwiązania. Otrzymany wynik jest jednak rozwiązaniem przybliżonym, o błędzie zależnym od wybranego rodzaju schematu różnicowego. Przybliżenie pochodnej różnicą centralną daje błąd o rząd mniejszy niż w przypadku różnicy w przód lub wstecznej, tzn. ma błąd obcięcia $\mathcal{O}(\Delta x)^2$. Fakt ten przemawia za użyciem jej w projekcie[8].

[8] **Anderson**; **s. 132**.

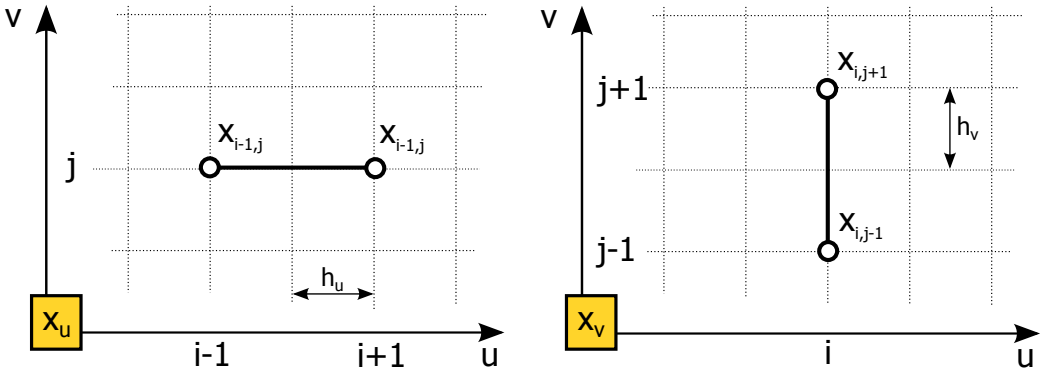


Rysunek 2.3: Interpretacja geometryczna różnic skończonych: przedniej, wstecznej oraz centralnej wyraźnie wskazuje na przewagę tej ostatniej pod względem wielkości błędu popełnianego przy dyskretyzacji pochodnej.

Poniżej użyto oznaczeń h_u, h_v na wartości kroku siatki odpowiednio na kierunku osi Ou oraz Ov.

Różnice dla pochodnych pierwszego rzędu:

$$x_u = \frac{x_{i+1,j} - x_{i-1,j}}{2h_u} \qquad x_v = \frac{x_{i,j+1} - x_{i,j-1}}{2h_v}$$



Rysunek 2.4: Konstrukcja różnic skończonych drugiego rzędu zmiennej $x = x(u, v)$

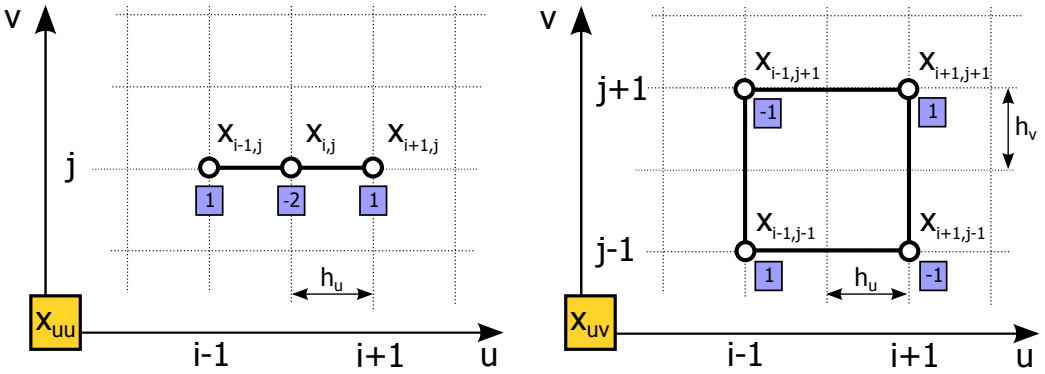
Konstrukcja różnic dla y_u, y_v jest analogiczna

$$y_u = \frac{y_{i+1,j} - y_{i-1,j}}{2h_u} \qquad y_v = \frac{y_{i,j+1} - y_{i,j-1}}{2h_v}$$

Różnice dla pochodnych drugiego rzędu (rys. 2.5):

$$x_{uu} = \frac{x_{i+1,j} - 2x_{i,j} + x_{i-1,j}}{h_u^2} \quad x_{uv} = \frac{x_{i+1,j+1} - x_{i+1,j-1} - x_{i-1,j+1} + x_{x-1,j-1}}{4h_u h_v}$$

$$y_{uu} = \frac{y_{i+1,j} - 2y_{i,j} + y_{i-1,j}}{h_u^2} \quad y_{uv} = \frac{y_{i+1,j+1} - y_{i+1,j-1} - y_{i-1,j+1} + y_{i-1,j-1}}{4h_u h_v}$$



Rysunek 2.5: Konstrukcja różnic skończonych drugiego rzędu dla zmiennej $x = x(u, v)$

Dla uproszczenia dalszego zapisu wprowadźmy następujące oznaczenie na wartość zmiennej ϕ w węźle o współrzędnych (u_i, v_i):

$$\phi(u_i, v_j) \triangleq \phi_{i,j}$$

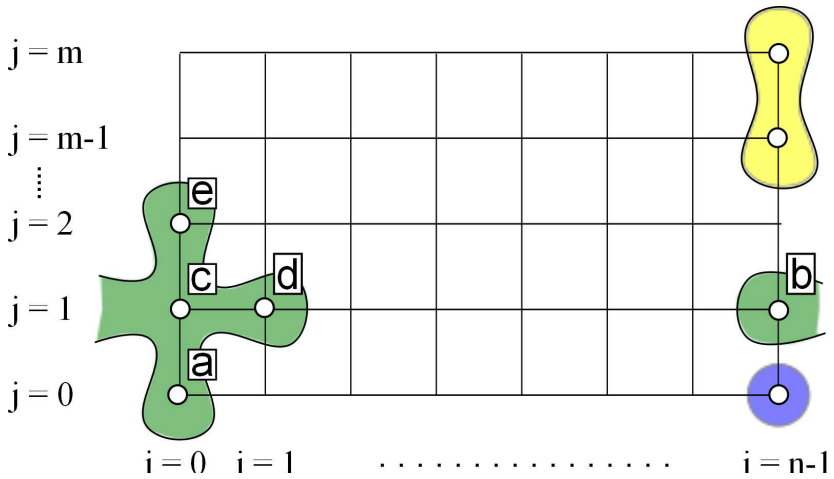
Wstawiając powyższe różnice do (2.7), oraz analogicznie konstruując różnice dla zmiennej ϕ, po wstawieniu otrzymanych związków do wzoru na laplasjan we współrzędnych obliczeniowych (2.5) otrzymujemy

$$A \cdot \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h_u^2} + B \cdot \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h_v^2} + C \cdot \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h_v} + D \cdot \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2h_v} = -1 \tag{2.8}$$

Ostatecznie po uporządkowaniu względem ϕ równanie Poissona w postaci różnicowej wygląda następująco:

$$\underbrace{\left(\frac{B}{h_v^2}-\frac{D}{2h_v}\right)}_{a}\phi_{i,j-1} + \underbrace{\left(\frac{A}{h_u^2}-\frac{C}{2h_u}\right)}_{b}\phi_{i-1,j} \underbrace{-2\left(\frac{A}{h_u^2}+\frac{B}{h_v^2}\right)}_{c}\phi_{i,j} + \\ + \underbrace{\left(\frac{A}{h_v^2}+\frac{C}{2h_u}\right)}_{d}\phi_{i+1,j} + \underbrace{\left(\frac{B}{h_v^2}+\frac{D}{2h_v}\right)}_{e}\phi_{i,j+1} = -1 \tag{2.9}$$

Przedstawienie go w takiej formie umożliwia łatwe zapisanie układu równań dla całego obszaru obliczeniowego. Wzajemne rozmieszczenie przestrzenne węzłów odpowiadających poszczególnym wartościom zmiennej ϕ, tworzących tzw. pięciopunktową gwiazdę różnicową (ang. *five-point-star*), zaprezentowano na rys. 2.6.



Rysunek 2.6: Siatka z wyróżnionymi węzłami, tworzącymi pięciopunktową gwiazdę różnicową oraz warunki brzegowe: Dirichleta i Neumanna.

2.5 Warunki brzegowe

Układ równań utworzony na podstawie wzoru (2.9) należy uzupełnić o poniższe warunki brzegowe (rys. 2.6).

1. **warunek ciągłości** - węzły leżące na lewej i prawej krawędzi siatki są w rzeczywistości węzłami sąsiadującymi ze sobą. W równaniu (2.9)

zapisanym dla węzła tego rodzaju wartość współczynnika b (lub odpowiednio d) pochodzi z węzła leżącego na samym końcu (początku) tego samego rzędu siatki.

2. **warunek Dirichleta** - zerowa wartość zmiennej ϕ na krawędzi profilu

$$\phi\Big|_{\partial S} = \phi_{i,0} = 0 \tag{2.10}$$

3. **warunek Neumanna** - zerowa wartość pochodnej ϕ w kierunku normalnym do zewnętrznego brzegu obszaru obliczeniowego

$$\frac{\partial \phi}{\partial n} = 0 \iff \frac{\phi_{i,m} - \phi_{i,m-1}}{h_v} = 0 \tag{2.11}$$

gdzie: m - indeks zewnętrznego rzędu siatki

2.6 Rozwiązanie zagadnienia

Rozwiązanie uprzednio zdefiniowanego zagadnienia sprowadza się do rozwiązania macierzowego układu równań o postaci

$$[\mathbb{K}]\{\phi\} = \{f\} \tag{2.12}$$

gdzie:

$\mathbb{K}$ - macierz współczynników

$\{\phi\}$ - wektor poszukiwanych wartości funkcji ϕ

$\{f\}$ - wektor prawej strony

W rozwiniętej formie równanie prezentuje się następująco (puste miejsca oznaczają zerowe elementy)

$$\begin{bmatrix}
1 & & & & & & & & & & & & & & \\
 & 1 & & & & & & & & & & & & & \\
 & & \ddots & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & \\
a & & & & c & d & & b & e & & & & & & \\
 & a & & & b & c & d & & & e & & & & & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & \ddots & & & & \\
 & & & a & & & b & c & d & & & e & & & \\
 & & & & a & d & & b & c & & & & e & & \\
 & & & & & -1 & & & & 1 & & & & & \\
 & & & & & & -1 & & & & 1 & & & & \\
 & & & & & & & \ddots & & & & \ddots & & & \\
 & & & & & & & & -1 & & & & 1 & &
\end{bmatrix}
\begin{bmatrix}
\phi_{0,0} \\ \phi_{1,0} \\ \vdots \\ \phi_{n-1,0} \\ \phi_{0,1} \\ \phi_{1,1} \\ \vdots \\ \phi_{n-2,1} \\ \phi_{n-1,1} \\ \phi_{0,m} \\ \phi_{1,m} \\ \vdots \\ \phi_{n-1,m}
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0
\end{bmatrix} \tag{2.13}$$

Macierz współczynników ma rozmiar $(n \cdot m) \times (n \cdot m)$, gdzie n - liczba węzłów na profilu, m - liczba rzędów siatki. Jej struktura jest charakterystyczna dla zagadnień opisanych równaniami różniczkowymi cząstkowymi, tzn. większość jej elementów jest zerowa. Macierze o powyższej własności nazywane są rzadkimi (ang. *sparse*)[9]. Dla rozważanego w niniejszym projekcie problemu rozmiar macierzy wynosi $N \times N = 12\,524 \times 12\,524 = 156\,850\,576$ elementów, z czego tylko 0,03% ma niezerowe wartości[10].

Do rozwiązania układu (2.13) zdecydowano się na skorzystanie ze stabilizowanej metody wzajemnie sprzężonych gradientów BiCGSTAB (ang. *Bi-Conjugate Gradient STABilized method*). Jest to metoda iteracyjna, oparta na podprzestrzeniach Kryłowa, odpowiednia do zagadnień dobrze uwarunkowanych, z macierzami rzadkimi[11]. Za jej wykorzystaniem przemawia również dostępność implementacji w bibliotece Eigen (patrz podrozdział 3.2).
Tolerancja rozwiązania została przyjęta na poziomie tolerancji danych wejściowych, tzn. $\epsilon = 10^{-6}$.

W wyniku rozwiązania układu (2.13) otrzymuje się wartości ϕ wyrażone w przestrzeni obliczeniowej. Aby móc skorzystać ze wzoru na odległość (2.3)

[9] **Saad**; **s. 67**.

[10] Macierz współczynników zawiera po n wartości wynikających z warunków brzegowych Dirichleta i Neumanna, oraz $5n(m-2)$ z zapisu różnicowego równania Poissona. Procentowe wypełnienie macierzy można obliczyć następująco: $\frac{5n(m-2)+n+n}{n^2m^2} \approx \frac{5}{nm}$

[11] **Blazek**; **s. 208**.

należy rozwiązać pomocniczy układ równań wynikający wprost z reguły łańcuchowej (2.14), mnożąc obie strony przez odwrotność macierzy Jacobiego.

$$\begin{cases} \phi_u = \phi_x \cdot x_u + \phi_y \cdot y_u \\ \phi_v = \phi_x \cdot x_v + \phi_y \cdot y_v \end{cases} \iff \begin{bmatrix} \phi_u \\ \phi_v \end{bmatrix} = \underbrace{\begin{bmatrix} x_u & y_u \\ x_v & y_v \end{bmatrix}}_{\text{macierz Jacobiego}} \begin{bmatrix} \phi_x \\ \phi_y \end{bmatrix} \tag{2.14}$$

$$\begin{bmatrix} \phi_x \\ \phi_y \end{bmatrix} = \begin{bmatrix} x_u & y_u \\ x_v & y_v \end{bmatrix}^{-1} \begin{bmatrix} \phi_u \\ \phi_v \end{bmatrix} \tag{2.15}$$

Szukana zależność transformacyjna (2.15) wymaga dyskretyzacji pochodnych cząstkowych według przedstawionego wcześniej schematu różnicowego.

$$\phi_u = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h_u} \quad \phi_v = \frac{\phi_{i,j} - \phi_{i,j-1}}{h_v}$$

Rozdział 3

Implementacja

3.1 Układ logiczny programu

Funkcjonalność programu została podzielona między klasy cGrid, cSolution oraz cSolver. Pierwsza z nich przechowuje wygenerowaną siatkę obliczeniową, druga otrzymane rozwiązanie, natomiast ostatnia z nich jest klasą nadrzędną, z jej poziomu wywoływane są wszystkie operacje przewidziane w funkcjonalności programu.

3.2 Biblioteki zewnętrzne

Do operacji na macierzach wykorzystano darmową bibliotekę języka C++ Eigen (wersja 3.1.2). Za jej wyborem do niniejszego projektu przemawia niezawodność i szybkość operacji na macierzach dowolnych rozmiarów, oraz zaimplementowanie metod do rozwiązania rzadkich układów macierzowych[1].

3.3 Klasa cGrid

Przechowuje węzły siatki oraz umożliwia do nich dostęp. Do przechowywania węzłów posłużono się kontenerem vector zaimplementowanym w bibliotece standardowej języka C++.

```
std::vector<cPoint> mPoints;
```

W przeciwieństwie do dwuwymiarowej struktury siatki sam kontener przechowuje elementy w porządku liniowym, co wymusza dodanie operatora dostępu do elementu (węzła) o indeksach (i, j).

```
cPoint & operator()(const int i, const int j)
```

Dodawanie węzłów do kontenera odbywa się przy pomocy metody

```
void     addNode(const cPoint & inPoint, eGridMode mode = GRID)
```

[1] http://eigen.tuxfamily.org/

gdzie oprócz referencji do węzła przekazuje się jego typ GRID / PROFILE. Jest to o tyle ważne, ponieważ dodanie węzła profilu zwiększa wartość wewnętrznego licznika, na podstawie którego jest generowana siatka i przeprowadzane rozwiązanie metodą siłową.
Poniższe metody dostępowe zwracają odpowiednio liczbę węzłów (GRID - siatki, PROFILE - na profilu) oraz rzędów siatki

```
int nodeCount(eGridMode mode = GRID) const
int rowCount() const;
```

3.4 Klasa cSolution

Publiczna klasa dziedziczona po klasie cGrid, oprócz obiektu siatki dodatkowo przechowuje w kontenerze typu vector wartości obliczonego rozwiązania zagadnienia. Poniższa metoda dodaje do kontenera wartość inValue typu (DISTANCE, PHI)

```
void    addValue(double inValue, eSolutionData valueType= DISTANCE);
```

Analogicznie zwrot zapisanych wartości uzyskuje się korzystając z metody

```
double  getValue(int k, eSolutionData valueType = DISTANCE);
```

gdzie k jest bezwzględnym numerem węzła

3.5 Klasa cSolver

Nadrzędna klasa programu będąca jego interfejsem. Przechowuje obiekt mSolution typu cSolution oraz obiekty klasy Eigen, które wykorzystywane są do rozwiązania układu macierzowego:

```
//Macierz współczynników
Eigen::SparseMatrix<double>      K;

//Wektor prawej strony
Eigen::VectorXd                  f;

//Wektor rozwiązania
Eigen::VectorXd                  x;

//Obiekt rozwiązujący układ
Eigen::BiCGSTAB<Eigen::SparseMatrix<double>>     solver;
```

3.5.1 Metody prywatne

Metoda implementująca rozwiązanie siłowe

```
void    solveBruteForce()
```

Metoda implementująca rozwiązanie równaniem Poissona

```
void    solvePoisson()
```

Metoda bezpośrednio wyznaczająca odległość między węzłami

```
double  distanceBetween(cPoint & p_1, cPoint & p_2)
```

- argumenty:
  p_1 - referencja do pierwszego węzła
  p_2 - referencja do drugiego węzła
- zwraca: odległość $d(p_1, p_2)$

Metoda wyznaczająca odległość na postawie wzoru (2.15)

```
void    computeDistance()
```

Metoda obliczająca pochodne cząstkowe dla danego węzła siatki

```
double  partial(int k, ePartial partial);
```

- argumenty:
  k - bezwzględny numer węzła
  partial - rodzaj pochodnej $(x_u, x_v, y_u, y_v, x_{uu}, x_{uv}, x_{vv}, y_{uu}, y_{uv}, y_{vv})$
- zwraca: wartość pochodnej

3.5.2 Metody publiczne

Metoda wczytująca do pamięci współrzędne węzłów profilu

```
void    addProfile(std::string filePath)
```

- argument: ścieżka do pliku z danymi wejściowymi

Metoda generująca siatkę obliczeniową wokół profilu

```
void    generateGrid(cPoint & circleCenter,
                     double circleRadius,
                     int linesDensity)
```

- argumenty:
  circleCenter - punkt środka siatki

circleRadius - promień siatki
linesDensity - ilość rzędów siatki

Metoda rozwiązująca zagadnienie

```
double   solve(eSolutionType type = POISSON)
```

- argumenty:
  type - rodzaj używanej metody (POISSON, BRUTE FORCE)
- zwraca: czas wykonania w sekundach

Metoda zapisująca rozwiązanie do pliku

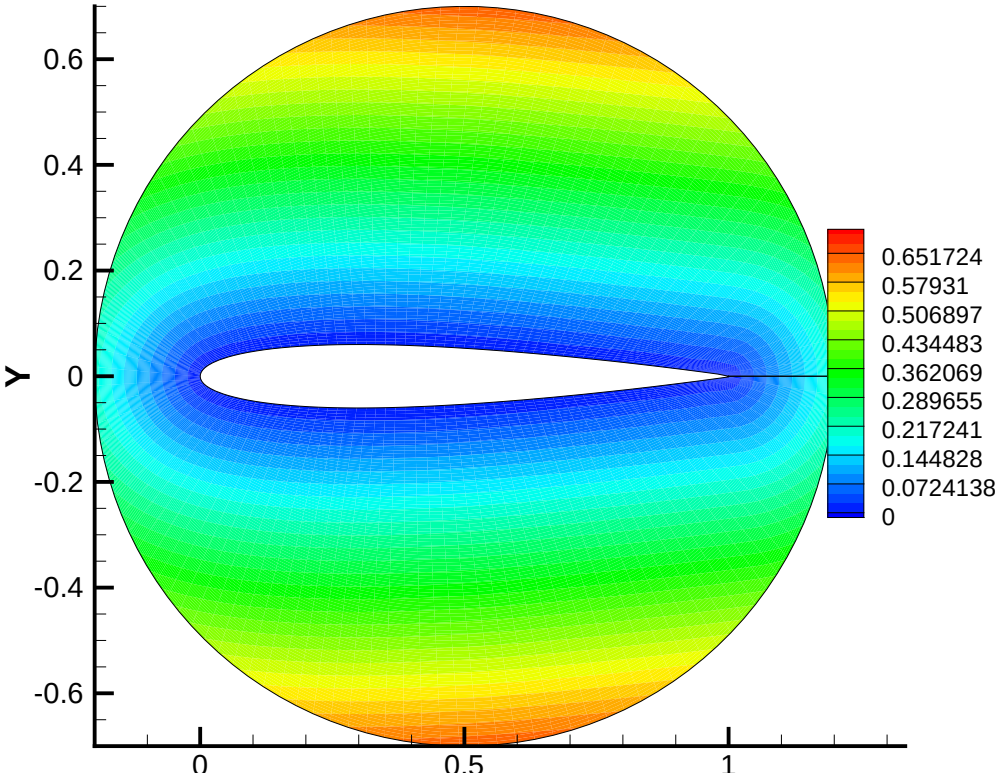
```
void     saveSolution(eSolutionData dataType = DISTANCE,
                      std::string filePath = "solution.dat",
                      eFileExtension extension = TECPLOT)
```

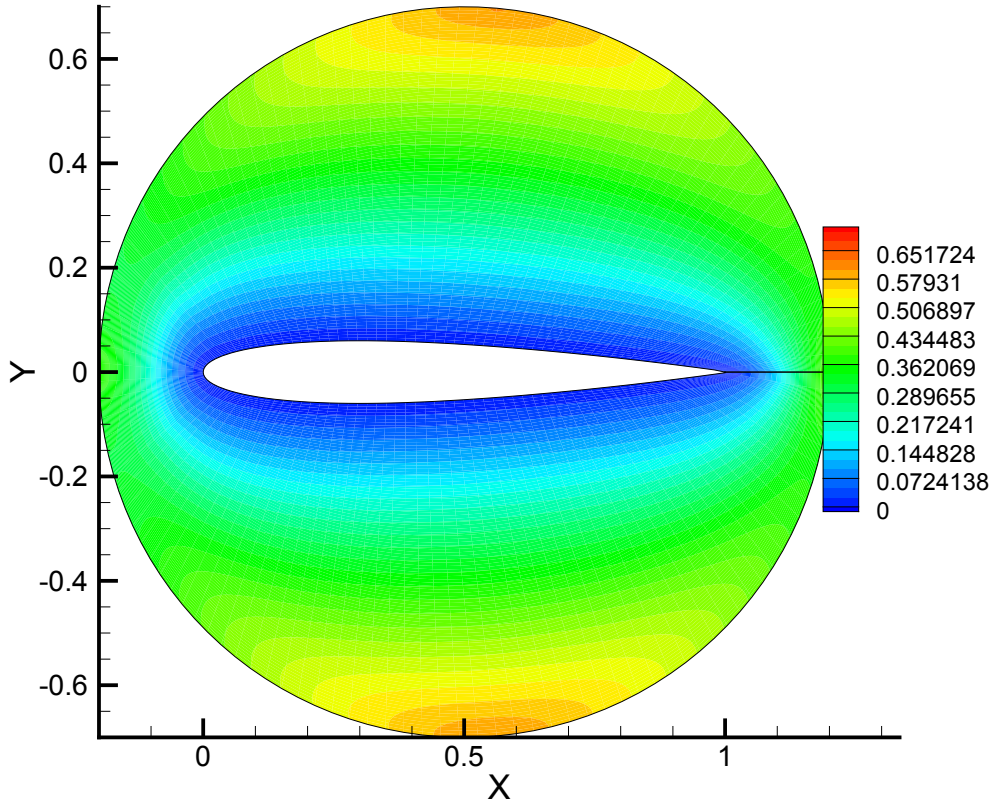
- argumenty:
  dataType - rodzaj zapisywanej danej, domyślnie odległość
  filePath - ścieżka i nazwa pliku wynikowego
  extension - opcjonalne dodanie nagłówka programu Tecplop

Rozdział 4

Wyniki

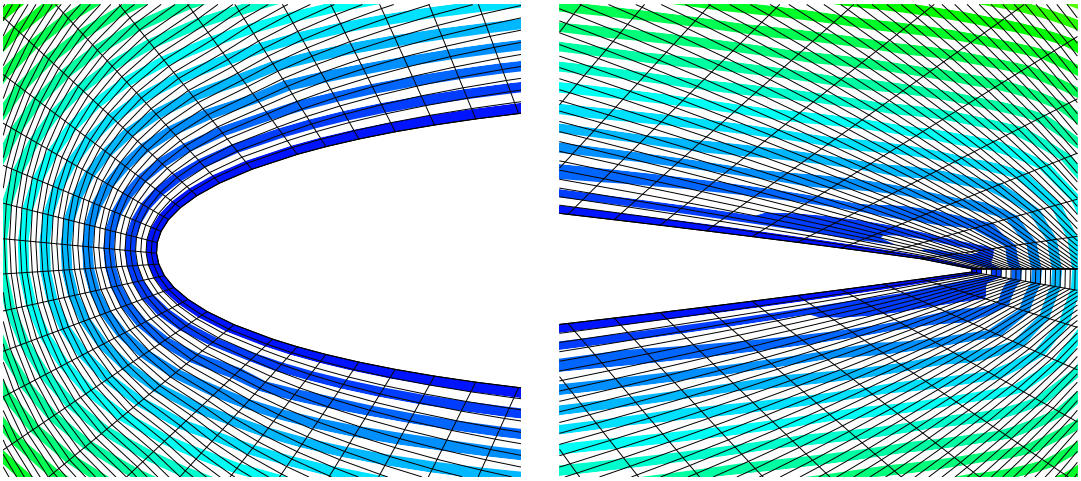
Przeprowadzono uruchomienie programu dla dwóch profili lotniczych: symetrycznego NACA 0012 oraz NACA 8207. Współrzędne punktów tworzących ich obrysy zaczerpnięto ze strony internetowej `http://www.ae.illinois.edu/m-selig/ads/coord_database.html`. Ze względu na użycie w projekcie naiwnego sposobu generowania siatki obliczeniowej, o jednorodnym odstępie między węzłami w kierunku v, zdecydowano się na zagęszczenie siatki do stu rzędów. Do celów porównawczych dla każdego z przypadków przeprowadzono obliczenia metodą siłową. Otrzymane pliki wyjściowe .dat wczytano do programu Tecplot, a wygenerowane mapy konturowe przedstawiono poniżej na rysunkach. Na końcu rozdziału zaprezentowano wykresy obliczonej odległości dla pierwszego rzędu siatki.



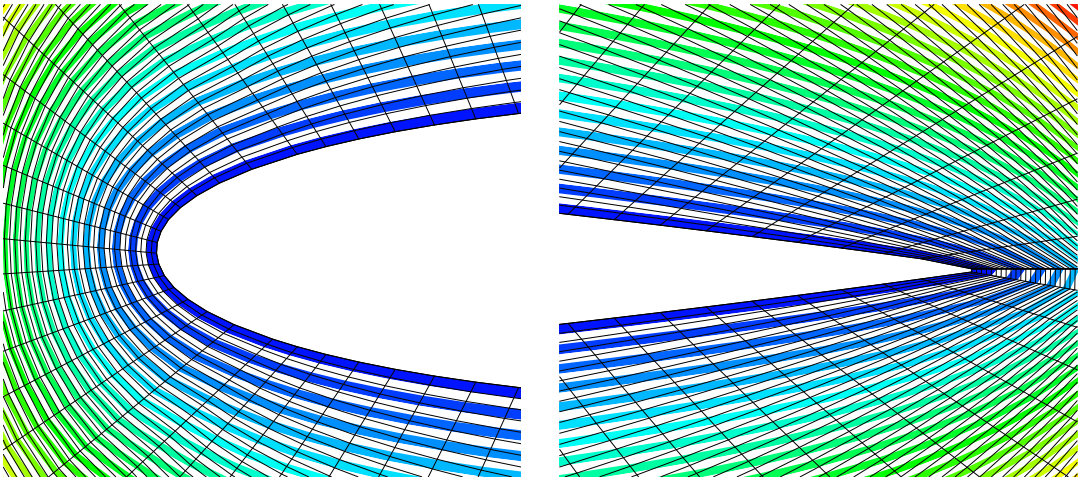


(b) Metoda Poissona

Rysunek 4.1: Profil NACA 0012

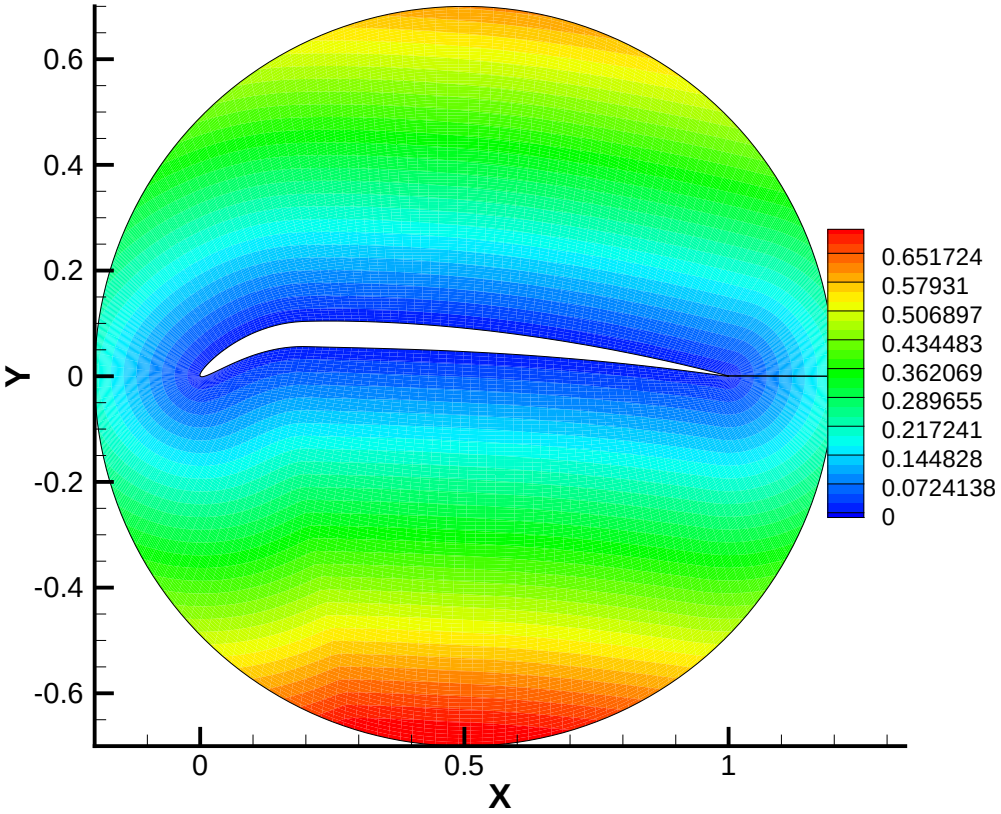


(a) Metoda siłowa

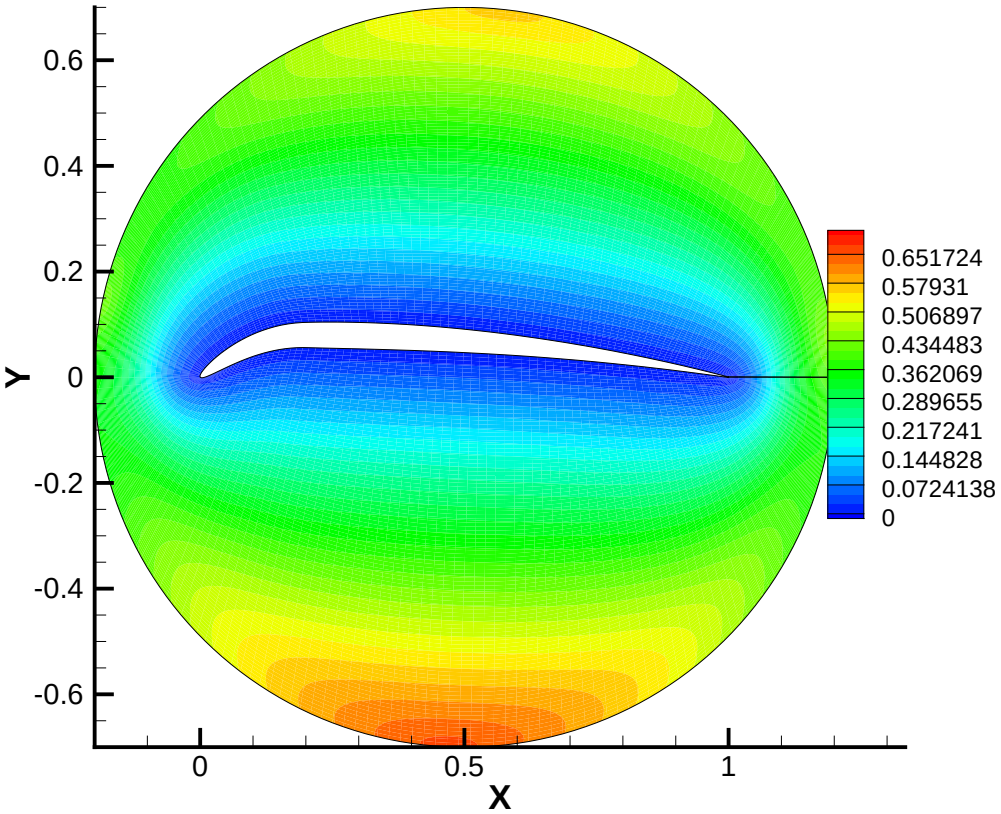


(b) Metoda Poissona

Rysunek 4.2: Profil NACA 0012 (po lewej nosek profilu, po prawej ostrze)

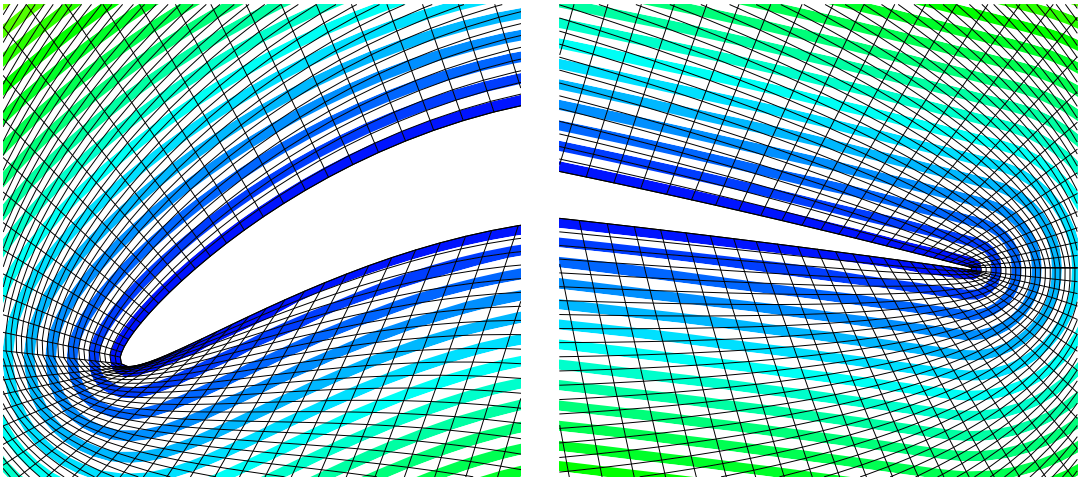


(a) Metoda siłowa

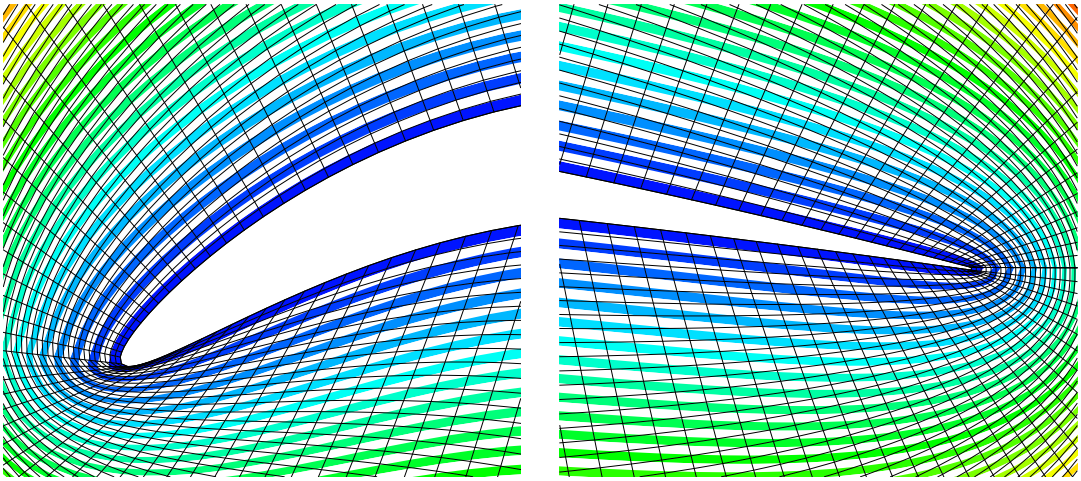


(b) Metoda Poissona

Rysunek 4.3: Profil NACA 8207

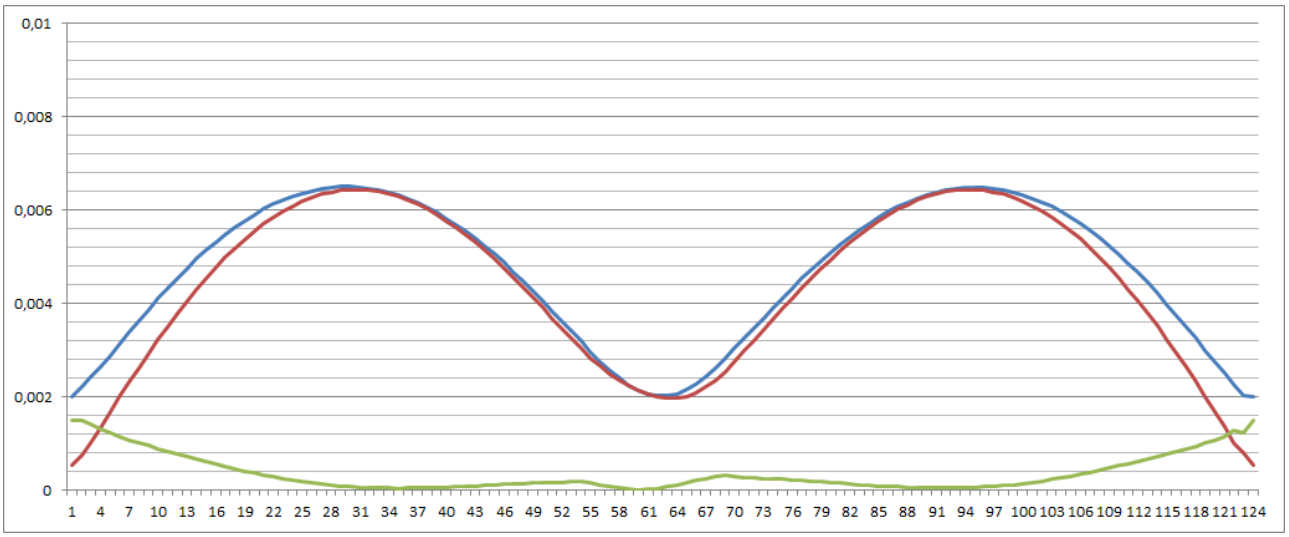


(a) Metoda siłowa

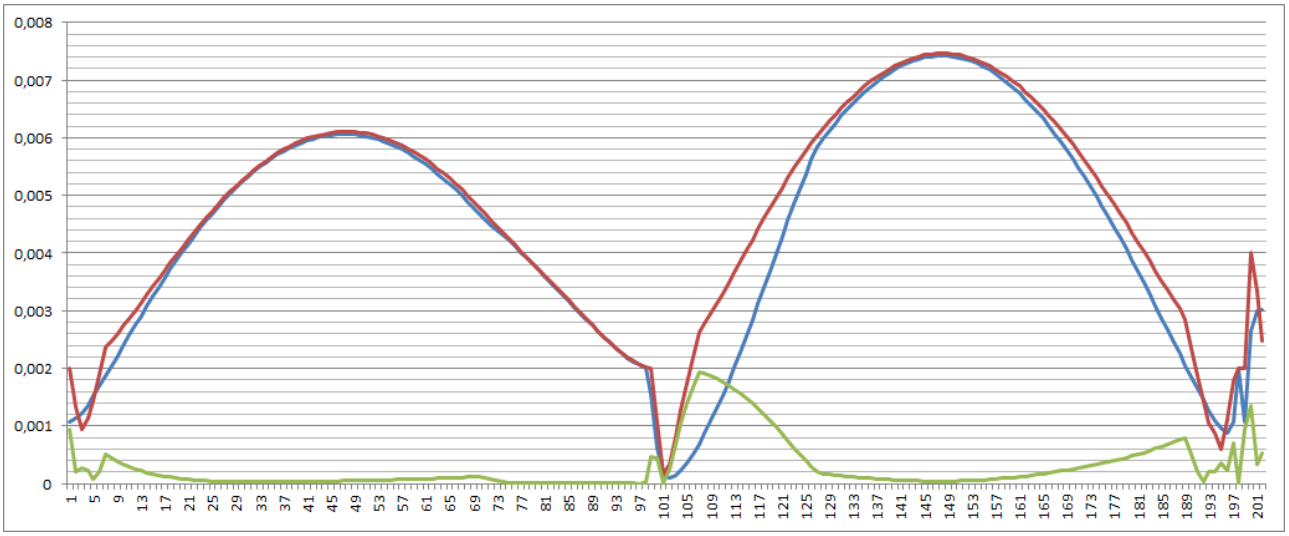


(b) Metoda Poissona

Rysunek 4.4: Profil NACA 8207 (po lewej nosek profilu, po prawej ostrze)



Rysunek 4.5: Wykres odległości dla pierwszego rzędu siatki dla metody Poissona i siłowej (NACA 0012). Kolorem zielonym oznaczono błąd bezwzględny.



Rysunek 4.6: Wykres odległości dla pierwszego rzędu siatki dla metody Poissona i siłowej (NACA 8207). Kolorem zielonym oznaczono błąd bezwzględny.

Rozdział 5

Wnioski

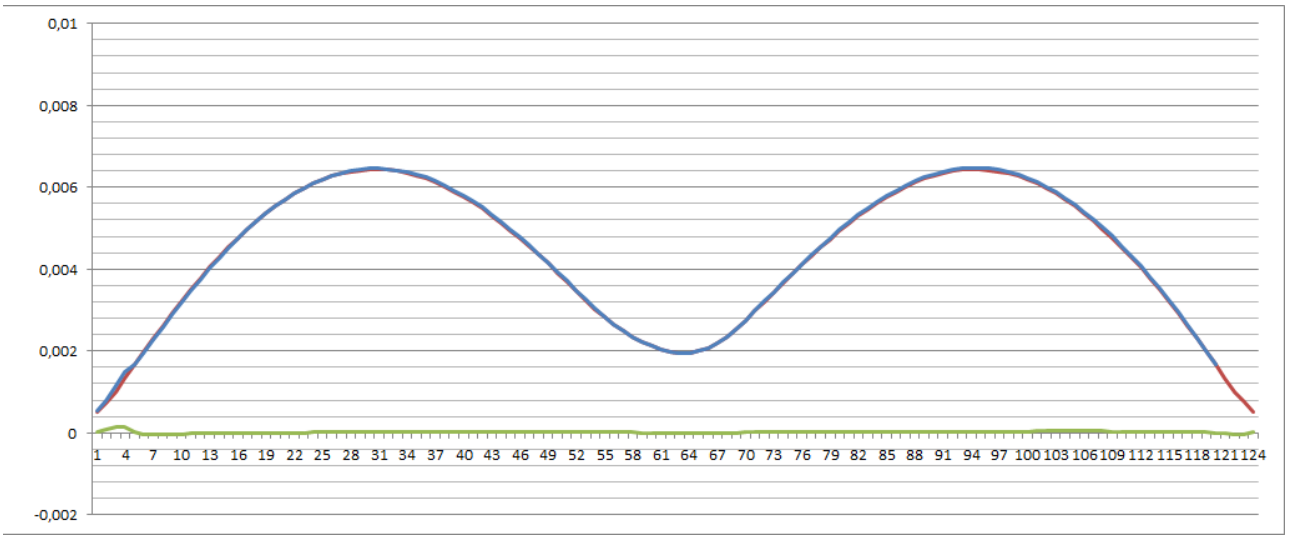
Z wykresu (4.5) wynika, że odległość od brzegu wyznaczona metodą Poissona nie odbiega znacznie od dokładnej wartości obliczonej metodą siłową. Błąd w rejonie ostrza profilu można powiązać ze słabą jakością utworzonej tam siatki. Jest to również widoczne w rejonie noska i ostrza dla profilu o wygiętej szkieletowej NACA 8207 (wykres 4.6).

Tabela 5.1: Porównanie czasów wykonania metody siłowej i Poissona dla profili użytych w projekcie

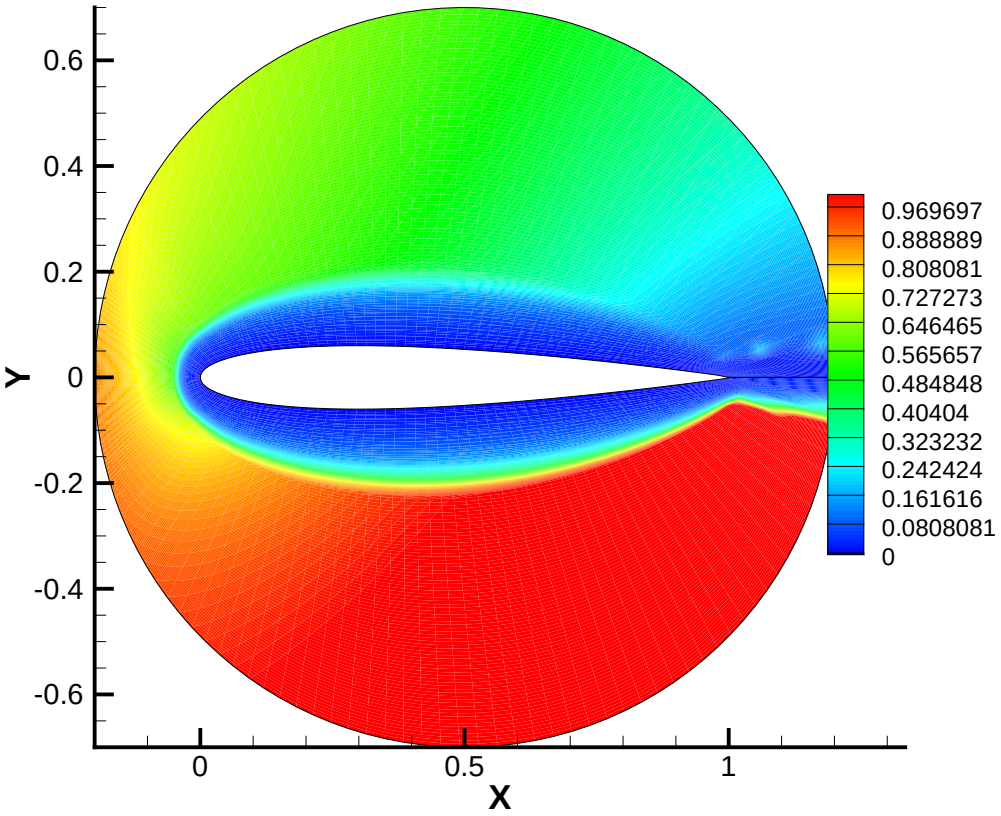
Metoda	NACA 0012	NACA 8207
Siłowa	0.8 s	2.1 s
Równanie Poissona	24 s	30 s
Ilość iteracji	208	168
Dokładność rozwiązania iteracyjnego	0,09	0,07
Równanie Poissona	Czas	Iteracje
1) zerowy wektor początkowy	24 s	208
2) dobrany wektor początkowy	2.7 s	14

Głównym mankamentem rozwiązania otrzymanego metodą Poissona jest czas wykonywania obliczeń, o rząd wielkości większy od rozwiązania siłowego (tab. 5.1). Remedium jest spostrzeżenie, iż interesującym nas obszarem dokładnego rozwiązania jest bliskie sąsiedztwo brzegu profilu. Jeżeli udałoby się rozwiązać układ macierzowy w mniejszej liczbie iteracji, nie pogarszając przy tym wyników blisko brzegu, to zaoszczędzilibyśmy znaczną ilość czasu obliczeniowego. Taka możliwość istnieje, jeżeli zdefiniujemy wektor początkowy dla algorytmu iteracyjnego, o wartościach "z grubsza" odpowiadających rozkładowi rozwiązania w przestrzeni fizycznej[1]. Czas obliczeń zbliża się wtedy do czasu wykonania metody siłowej (tab. 5.1), a rozwiązanie w interesującym nas obszarze pozostaje poprawne (wykres 5.1).

[1]W programie zostało to zaimplementowane jako zainicjalizowanie zmiennej ϕ_i wartością $i \cdot mod(n) \cdot \Delta$



Rysunek 5.1: Wykres odległości dla pierwszego rzędu siatki dla metody Poissona z dobranym wektorem początkowym i zerowym (NACA 0012). Kolorem zielonym oznaczono błąd bezwzględny.



Rysunek 5.2: Rozwiązanie metodą Poissona otrzymane przy wykorzystaniu wektora początkowego

5.1 Propozycje dalszego rozwoju

1. Jak wspomniano wcześniej, wybór iteracyjnej metody rozwiązania BiCSTAB związany był z dostępnością jej implementacji w bibliotece Eigen. Istnieją jednak dedykowane metody rozwiązania równania Poissona, które mogą przyspieszyć czas wykonywania programu. Jedną z nich jest Fast Poisson Solver, szerzej opisany w książce Saada[2].

2. W oparciu o istniejącą architekturę programu należałoby dodać moduł rozwiązujący zagadnienie wykorzystując równanie Eikonału. Jest to równanie różniczkowe cząstkowe typu hiperbolicznego, dodatkowo nieliniowe. Implementacja będzie wymagać odmiennego modelu matematycznego oraz sposobu rozwiązania.

3. Zamiast generować siatkę w programie, można ją otrzymać korzystając z pomocy zewnętrznego oprogramowania do preprocesingu (np. Gambit)

[2] **Saad**; **s. 58**.

Bibliografia

[1] Anderson J.D., Computational Fluid Dynamics. The basics with applications, McGraw-Hill, 1995.

[2] Blazek J., Computational Fluid Dynamics: Principles and Applications, Elsevier, 2001.

[3] Saad Y. Iterative methods for sparse linear systems. Second edition. SIAM, 2003.

[4] Sethain J.A. Level set methods and fast marching methods. Cambridge University Press, 1999.

[5] Tucker P.G., Hybrid Hamilton-Jacobi-Poisson wall distance function model. Computers & Fluids 44, (130-142), 2011.

Dodatek A

Opis interfejsu programu

Niniejszy dodatek opisuje sposób korzystania ze stworzonego w ramach niniejszego projektu programu wallDistanceSolver.exe, oraz jego interfejsu, do własnego użytku.

A.1 Dane wejściowe

Danymi wejściowymi do programu są współrzędne profilu zapisane w dwóch kolumnach, kolejno dla kierunku x oraz y, z użyciem kropki jako separatora dziesiętnego (kod źródłowy A.1). Domyślna nazwa pliku to `NACA_0012.dat`

Kod źródłowy A.1 : Format zapisu danych wejściowych

```
1   0.964244    0.006169
    0.947231    0.008434
3   ........    ........
    ........    ........
```

Sugeruje się nie pozostawiać pustych wierszy, złamanych znakiem nowej linii (enterem) na końcu kolumn, aby uniknąć możliwego wczytania nic nie znaczących znaków. Należy również zwrócić uwagę, aby wartości w wierszach nie powtarzały się, bowiem spowoduje to utworzenie kilku węzłów w tym samym punkcie, co przełoży się na rozwiązanie błędnie wygenerowanego układu równań (2.13).

A.2 Dane wyjściowe

Danymi wyjściowymi z programu są współrzędne (x, y) węzłów siatki oraz wartość odległości od brzegu d, zapisane w trzech kolumnach w kolejności jak powyżej, z użyciem kropki jako separatora dziesiętnego (kod źródłowy A.2). Dodatkowo program umożliwia wstawienie nagłówka niezbędnego do wczytania danych do oprogramowania `Tecplot 360`[1]

[1] `ftp://ftp.tecplot.com/pub/doc/tecplot/360/dataformat.pdf`, s. 134

Kod źródłowy A.2 : Format zapisu danych wyjściowych

Zwykłe formatowanie

```
                1              0              0
2       0.979641       0.004079              0
        0.964244       0.006169              0
4       ........       ........       ........
        0.850307       0.032659       0.012574
6       0.831428       0.035881       0.013543
        ........       ........       ........
```

Format danych programu Tecplot 360

```
1   VARIABLES = "X", "Y", "U",
    ZONE I=124, J=41, DATAPACKING=POINT
3
                1              0              0
5       0.979641       0.004079              0
        0.964244       0.006169              0
7       ........       ........       ........
        0.850307       0.032659       0.012574
9       0.831428       0.035881       0.013543
        ........       ........       ........
```

A.3 Przykładowy program

W pierwszym kroku definiowane są zmienne pomocnicze, określające promień siatki circleRadius, liczbę rzędów gridDensity oraz położenie jej środka S (korzystając z klasy pomocniczej cPoint).

Następnie następuje inicjalizacja obiektu solvera CASE, oraz wczytanie z pliku współrzędnych punktów profilu metodą addProfile(). Na podstawie tych danych tworzona jest siatka o wcześniej zdefiniowanych parametrach (metoda generateGrid()). Rozwiązanie zagadnienia odbywa się przy użyciu metody solve(POISSON)[2]

W ostatnim kroku otrzymane rozwiązanie jest zapisywane metodą saveSolution(). Treść komunikatów programu przedstawiono w kodzie źródłowym (A.4).

[2]Dla celów porównawczych można skorzystać z metody siłowej. W tym celu należy zmienić typ wyliczeniowy z POISSON na BRUTE_FORCE.

Kod źródłowy A.3 : Przykładowy program rozwiązujący zagadnienie

```
#include <iostream>
#include "cSolver.h"

int main(int argc, char * argv[]){
    //Grid parametres
    cPoint  S              = cPoint(0.5, 0.0);
    double  circleRadius   = 1;
    int     gridDensity    = 40;

    cSolver CASE;

    //Read profile from a file
    CASE.addProfile("../../data/NACA_0012.dat");

    //Generate grid based on profile
    CASE.generateGrid(S, circleRadius, gridDensity);

    //Solve problem
    std::cout << CASE.solve(POISSON) << std::endl;

    //Save solution to a file
    CASE.saveSolution(DISTANCE);

    return 0;
}
```

Kod źródłowy A.4 : Widok konsoli programu

```
Generating grid
Grid generated!
Solving Poisson...
    #iterations: 81 estimated error: 1.83435e-06
Solution solved!
6.908
Saving solution...
Solution saved!
```

Dodatek B

Zawartość płyty CD

Pliki źródłowe programu (folder Source files)

- main.cpp
- cPoint.cpp
- cSolution.cpp
- cSolver.cpp
- cGrid.cpp

Pliki nagłówkowe (folder Source files)

- cPoint.h
- cSolution.h
- cSolver.h
- cGrid.h

Przykładowy program (folder Source files)

- wallDistanceSolver.exe

Pliki wejściowe (folder Data)

- NACA_0012.dat - współrzędne x,y profilu NACA 0012
- NACA_8207.dat - współrzędne x,y profilu NACA 8207

Dokumentacja

- Dokumentacja.pdf - wersja cyfrowa niniejszego opracowania