BACHELORARBEIT

Implementation und Anwendung eines Hess-Smith Panelverfahrens in Python

ausgeführt am Institut für Strömungsmechanik und Wärmeübertragung
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Braun

durch

Leon Schwarzäugl

Gumpendorfer Straße 137,
1060 Wien

6. August 2022

Zusammenfassung

Im Zuge der vorliegenden Arbeit wurde ein Panelverfahren nach Hess-Smith zur Berechnung von ebenen Potentialströmungen um geschlossene Körper implementiert und auf verschiedene Probleme angewandt. Dazu wurde die Programmiersprache Python verwendet.

Zunächst wurde eine Klasse für Panele und ihre Eigenschaften entworfen. Im Zuge dessen wurde eine Routine entwickelt, die die entsprechenden Parameter in eine .csv-Datei ausgibt. Danach wurde eine Klasse für Tragflächenprofile sowie Methoden zur Berechnung der ihnen zugehörigen Systemparameter implementiert. Mit den somit berechneten Systemparametern kann das sich ergebende lineare Gleichungssystem für die Quellstärken gelöst werden. Im Anschluss wurden aus den ermittelten Quellstärken die Tangentialgeschwindigkeiten und Druckbeiwerte an den jeweiligen Panelen, sowie unter Hinzunahme einer Wirbelbewegung der Auftriebsbeiwert für das Gesamtsystem ermittelt. Abschließend wurde eine Routine für die Ermittlung der Tangentialgeschwindigkeiten an beliebigen Punkten im Strömungsfeld entwickelt.

Die fertige Implementation wurde verwendet, um für die Panele verschiedener Tragflächenprofile Graphen zu Geometrien und Parametern zu gewinnen. Anschließend wurde anhand eines Kreiszylinders mit 8 Panelen gleicher Seitenlänge der Vektor der Quellbelegung, sowie die Tangentialgeschwindigkeiten und Druckbeiwerte an den Panelmittelpunkten ermittelt. Die Ergebnisse für die Druckbeiwerte wurden mit dem theoretischen Ergebnis aus der Potentialtheorie verglichen. Dabei wurde für die mittlere Abweichung in Abhängigkeit der Panelanzahl n eine Abhängigkeit $\propto 1/n^2$ ermittelt. Abschließend wurde das oben genannte System um eine Wirbelbelegung erweitert und der Auftriebsbeiwert unter verschiedenen Anstellwinkeln des Profils bei konstantem Angriffswinkel berechnet.

Schließlich wurde die Implementierung auf Joukowski-Profile angewandt und eine experimentelle Abschätzung der Fehlerordnung des Hess-Smith-Verfahrens durchgeführt. Dabei ergab sich eine relative Abweichung des Auftriebsbeiwerts des Hess-Smith Verfahrens vom theoretischen Resultat von 3.3%.

Inhaltsverzeichnis

1 Aufgabenstellung **1**
1.1 Zielsetzung . . . 1
1.2 Methodik . . . 1

2 Grundlagen **2**
2.1 Hess-Smith-Panelverfahren . . . 2
2.1.1 Berücksichtigung einer Wirbelbelegung . . . 5
2.1.2 Berechnung der Systemmatrix . . . 6

3 Bericht der Arbeit **8**
3.1 Vorbereitung der Daten . . . 8
3.2 Untersuchungen am Kreiszylinder . . . 10
3.2.1 Achtseitiger Kreiszylinder . . . 11
3.2.2 Untersuchungen an allgemeinen Kreiszylindern . . . 12
3.3 Auswertung ausgewählter Profile . . . 15
3.3.1 NACA0012-Profil . . . 16
3.3.2 NASA SC(2)-0614-Profil . . . 17
3.3.3 Taylor Zone-40-Profil . . . 18
3.4 Abschätzung der Fehlerordnung anhand des Joukowski-Profils . . . 19

Literatur **i**
Anhang A: Lednicer- und Selig-Format . . . ii
Anhang B: Beispielausgabe der Methode .write_panels() . . . iii
Anhang C: Codeausschnitte . . . iv

1 Aufgabenstellung

1.1 Zielsetzung

Das Ziel der vorliegenden Arbeit ist die Berechnung aerodynamischer Eigenschaften von Tragflächenprofilen. Dazu sollen folgende Punkte implementiert werden:

- Eine Klasse für Panele und die sie definierenden Eigenschaften: Mittelpunkte auf der x- und y-Achse, Neigungswinkel θ_i und Länge l_i.
- Eine Klasse für Trägerprofile, ihre Umfänge U und Tiefe t sowie mathematisch relevante Systemparameter $\xi_{i,j}$, $\eta_{i,j}$, $I_{i,j}$, $J_{i,j}$, $A^{(n)}_{i,j}$, $A^{(t)}_{i,j}$, $M_{i,j}$.
- Eine Methode zur Lösung des linearen Gleichungssystems $M\vec{q} = \vec{b}$ für den Vektor der Quellbelegung unter einer konstanten Anströmgeschwindigkeit V_∞ und einem Anstellwinkel des Profils α, sowie der Berechnung der daraus resultierenden Tangentialgeschwindigkeiten und Druckbeiwerte.

Weiters soll eine Untersuchung der Abweichung der ermittelten Druckbeiwerte von den aus der Potentialtheorie berechneten Werten für einen achtseitigen Kreiszylinder sowie die Abweichung des Auftriebsbeiwerts eines Joukowski-Profils vom analytischen Resultat erfolgen.

1.2 Methodik

Zur Lösung der Aufgabenstellung wurde die Programmiersprache Python (Version 3.8) verwendet. [10]
Für die Lösung der linearen Gleichungssysteme wurde linalg.solve[1] aus der Python-Bibliothek NumPy (Version 1.23.1) verwendet. [6] Sämtliche Graphen wurden mithilfe der Python-Bibliothek Matplotlib (Version 3.5.2) erstellt. [8] Analytische Lösungen zu den linearen Gleichungssystemen wurden mit der Python-Bibliothek SymPy (Version 1.10.1) berechnet. [9] Für die Kurvenanpassung in 3.2.2 wurde die Python-Bibliothek SciPy (Version 1.8.1) verwendet. [11] Die Maskierung der Trägerprofile für die Strömungs- und Konturgraphen in 3.3 wurde mithilfe der Python-Bibliothek shapely (Version 1.8.2) erreicht. [5]

[1] linalg.solve verwendet die LAPACK routine _gesv als Solver [6]

2 Grundlagen

2.1 Hess-Smith-Panelverfahren

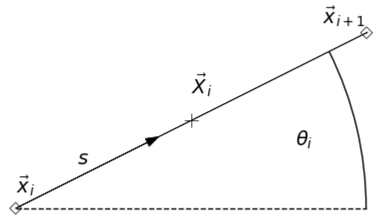


Abbildung 2.1: Zur Definition eines Panels

Das Hess-Smith-Panelverfahren ist ein Verfahren zur Berechnung von ebenen Potentialströmungen um geschlossene Körper mit Auftrieb. Ein kontinuierlicher zweidimensionaler Körper mit der Umfangkurve $\mathcal{C}$ wird dabei zunächst durch $n+1$ diskrete Datenpunkte $\vec{x}_i = (x_i, y_i)$ dargestellt. Die Kontur des Profils wird nun durch eine endliche Anzahl Verbindungsgeraden zwischen zwei benachbarten Punkten $\vec{x}_i, \vec{x}_{i+1}$, den Panelen $\mathcal{C}_i$ (Abb. 2.1) angenähert. Die Zählrichtung der Punkte sei als gegen den Uhrzeigersinn festgelegt (siehe Abb. 2.2). Für das Profil gilt somit die Periodizität $\vec{x}_n = \vec{x}_0$.
Jedes Panel wird durch die folgenden Parameter charakterisiert:

- den Mittelpunkten auf beiden Achsen

$$X_i = \frac{x_{n-i} + x_{n-i-1}}{2},\ Y_i = \frac{y_{n-i} + y_{n-i-1}}{2}, \tag{2.1}$$

- seinem Neigungswinkel

$$\theta_i = \left(\frac{y_{n-i-1} - y_{n-i}}{x_{n-i-1} - x_{n-i}} \right), \tag{2.2}$$

- und seiner Länge

$$l_i = \sqrt{(x_{n-i-1} - x_{n-i})^2 + (y_{n-i-1} - y_{n-i})^2}. \tag{2.3}$$

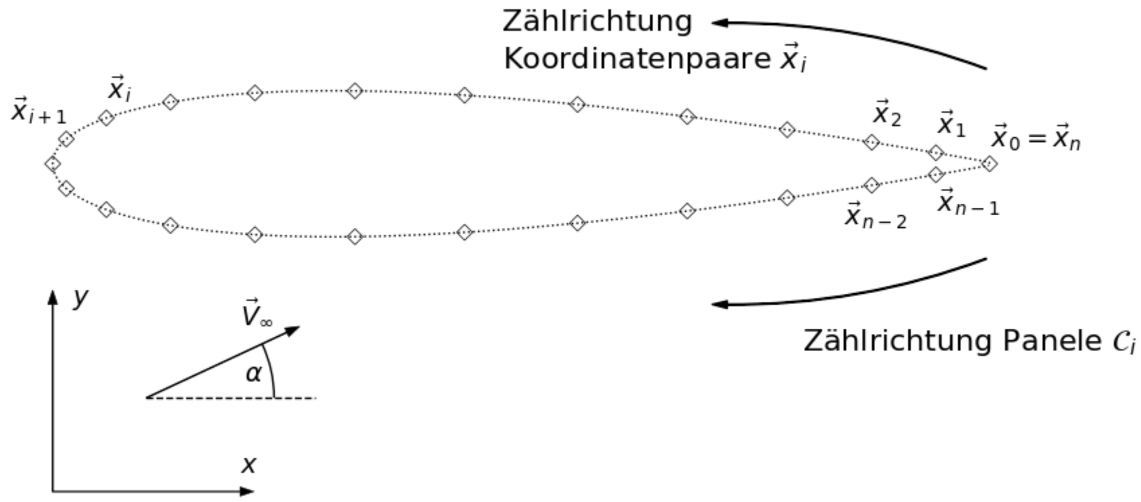


Abbildung 2.2: Zur Definition der Zählrichtung für Koordinatenpaare $\vec{x} = (x, y)$ und Panele $\mathcal{C}_i$ sowie der Anströmgeschwindigkeit V_∞ und Anströmungswinkel α

Die Indexierung der Panele verläuft somit im Uhrzeigersinn.
Im Hess-Smith-Panelverfahren sei nun jedes dieser Panele mit einer vorerst unbestimmten, konstanten Quelldichte q_i behaftet, welche an einem Punkt $\vec{r}_i = \vec{x} - \hat{x}_i(s)$ zum Geschwindigkeitspotential den Beitrag

$$\phi_i^{(q)} = \frac{q_i}{2\pi} \int_{\mathcal{C}_\rangle} ds \ln r_i \tag{2.4}$$

liefert, mit s, der Koordinate der Bogenlänge, welche das Profil parametrisiert.
Ebenso liefert die Profilanströmung mit der konstanten Geschwindigkeit V_∞ unter dem Winkel α ein Potential

$$\phi_\infty = V_\infty(\cos\alpha x + \sin\alpha y). \tag{2.5}$$

Aus der Summe über alle Panelbeiträge aus (2.4) und (2.5) ergibt sich das Gesamtgeschwindigkeitspotential zu

$$\phi(\vec{x}) = V_\infty(\cos\alpha x + \sin\alpha y) + \sum_{i=0}^{n-1} \phi_i^{(q)}. \tag{2.6}$$

Die Geschwindigkeit im Punkt $\vec{x}$ kann über den Gradienten von ϕ berechnet werden:

$$\vec{v}(\vec{x}) = \nabla\phi(\vec{x}). \tag{2.7}$$

Auf der Oberfläche des Profils muss die Strömung die Gleitbedingung erfüllen; damit lassen sich n Gleichungen aufstellen, um diese Bedingung zu modellieren. Es wird gefordert, dass die Normalkomponente des Geschwindigkeitsvektors,

$$v_j^{(n)} = \vec{v}(\vec{X}_j) \cdot \vec{n}_j, \tag{2.8}$$

verschwindet. Dabei ist $\vec{n}_j$ der Normalvektor auf das Panel $\mathcal{C}_j$.
In 0. Ordnung erhält man ein System von n Gleichungen

$$\sum_{j=0}^{n-1} M_{i,j}^{(q)} q_j = b_i, \tag{2.9}$$

mit

$$M_{i,j}^{(q)} = \frac{1}{q_j} \nabla \phi_j^{(q)}(\vec{X}) \cdot \vec{n}_i, \tag{2.10}$$

und den Inhomogenitäten

$$b_i = -V_\infty \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} \cdot \vec{n}_i, \tag{2.11}$$

aus dessen Lösung die Quellstärken q_i für jedes Panel ermittelt werden können. Damit können die Tangentialkomponenten des Geschwindigkeitsvektors

$$v_j^{(t)} = \vec{v}(X_j) \cdot \frac{\vec{x}_{j+1} - \vec{x}_j}{|\vec{x}_{j+1} - \vec{x}_j|}, \tag{2.12}$$

bestimmt werden.
Aus der Bernoulli-Gleichung

$$p_\infty + \frac{1}{2}\rho V_\infty^2 = p + \frac{1}{2}\rho v^2, \tag{2.13}$$

definiert man den Druckbeiwert als das Verhältnis der Druckdifferenzen von Profilanströmung und dynamischem Druck

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho V_\infty^2}. \tag{2.14}$$

Daraus kann der Druckbeiwert

$$c_{p_j} = 1 - \left(\frac{v_j^{(t)}}{V_\infty} \right) \tag{2.15}$$

am Mittelpunkt des Panels $\mathcal{C}_j$ bestimmt werden. [7] [4]

2.1.1 Berücksichtigung einer Wirbelbelegung

Will man den Auftrieb berücksichtigen, so reicht die reine Quellbelegung des Profils noch nicht aus.
Im Hess-Smith-Verfahren wird für das gesamte Profil eine einzige konstante Wirbelbelegung γ für alle Panele angenommen. Damit wird das Potential aus (2.6) um folgende Beiträge erweitert:

$$\phi_i^{(w)} = \frac{\gamma}{2\pi} \int ds\theta_i, \tag{2.16}$$

zu

$$\begin{aligned} \phi(\vec{x}) &= V_\infty(\cos\alpha x + \sin\alpha y) + \sum_{i=0}^{n-1} \left(\phi_i^{(q)} + \phi_i^{(w)}\right) \\ &= V_\infty(\cos\alpha x + \sin\alpha y) + \sum_{i=0}^{n-1} \int_{\mathcal{C}_i} \left(\frac{q_i}{2\pi} \ln r_i - \frac{\gamma}{2\pi}\theta_i\right) ds. \end{aligned} \tag{2.17}$$

Die Gleitbedingung bleibt erhalten, allerdings muss das Gleichungssystem entsprechend um den hinzugekommenen Geschwindigkeitsbeitrag erweitert werden. Dadurch erhält das Gleichungssystem eine neue Variable γ und ist somit unterbestimmt. Die fehlende Gleichung ergibt sich aus der Kutta-Bedingung, welche besagt, dass es an der Hinterkante des Profils keine Umströmung gibt.
Zur Modellierung der Kutta-Bedingung wählt man

$$v_0^{(t)} = -v_n^{(t)}. \tag{2.18}$$

Daraus folgt, dass die Tangentialkomponenten des Geschwindigkeitsvektors in den Mittelpunkten der Panele, welche die Profilhinterkante bilden, vom Betrag gleich groß und gegenläufig gerichtet sein müssen
Nach Lösung des erhaltenen linearen Gleichungssystems mit $q_n = \gamma$ kann nun der Auftriebsbeiwert c_a ermittelt werden. Dieser ergibt sich mit der Proftiltiefe t und der Länge l_i des Panels $\mathcal{C}_i$, approximiert in folgender Form:

$$c_a \approx \frac{2}{V_{\infty t}} \sum_{i=0}^{n-1} v_i^{(t)} l_i. \tag{2.19}$$

Für eine analytische Berechnung des Auftriebsbeiwerts muss nun noch die Systemmatrix bestimmt werden, welche das lineare Gleichungssystem (2.9) definiert. [7] [4] [2]

2.1.2 Berechnung der Systemmatrix

Die Systemmatrix zur Berechnung der Normalgeschwindigkeiten $v_i^{(n)}$ in den Panelmittelpunkten für eine stückweise konstante Quellenbelegung der Panel ergibt sich aus der Gleitbedingung. Wendet man diese direkt auf (2.6) an, muss ein kompliziertes Integral gelöst werden. Dieses kann theoretisch direkt mit Computermethoden gelöst werden, für den Berechnungsaufwand ist es allerdings bei Profilen mit einer hohen Anzahl an Panelen günstiger, dieses analytisch zu berechnen.
Für die Systemmatrix und ihre Parameter gelten:

$$M_{i,j} = A_{i,j}^{(n)}, \quad i = 0, 1, \dots, n-1; \quad j = 0, 1, \dots, n-1, \tag{2.20}$$

mit der Einflussmatrix

$$A_{i,j}^{(n)} = -\sin(\theta_i - \theta_j) I_{i,j} + \cos(\theta_i - \theta_j) J_{i,j}, \tag{2.21}$$

sowie den geometrischen Integralen

$$I_{i,j} = \begin{cases} \frac{1}{4\pi} \ln\left[\frac{(l_j + 2\xi_{i,j})^2 + 4\eta_{i,j}^2}{(l_j - 2\xi_{i,j})^2 + 4\eta_{i,j}^2}\right] & i \neq j \\ 0 & i = j \end{cases}, \tag{2.22}$$

und

$$J_{i,j} = \begin{cases} \frac{1}{2\pi} \arctan\left[\frac{l_j - 2\xi_{i,j}}{2\eta_{i,j}}\right] + \frac{1}{2\pi} \arctan\left[\frac{l_j + 2\xi_{i,j}}{2\eta_{i,j}}\right] & i \neq j \\ \frac{1}{2} & i = j \end{cases}. \tag{2.23}$$

Dabei gilt

$$\xi_{i,j} = (X_i - X_j)\cos\theta_j + (Y_i - Y_j)\sin\theta_j; \tag{2.24}$$

$$\eta_{i,j} = -(X_i - X_j)\sin\theta_j + (Y_i - Y_j)\cos\theta_j. \tag{2.25}$$

Mit der Systemmatrix und den Inhomogenitäten b_i wird nun das lineare Gleichungssystem aus (2.9) berechnet.
Für die Tangentialkomponente der Geschwindigkeit ergibt sich der Ausdruck

$$v_i^{(t)} = \sum_{j=0}^{n-1} A_{i,j}^{(t)} q_j + V_\infty \cos(\alpha - \theta_i), \tag{2.26}$$

mit der Einflussmatrix

$$A_{i,j}^{(t)} = \cos(\theta_i - \theta_j) I_{i,j} + \sin(\theta_i - \theta_j) J_{i,j}. \tag{2.27}$$

Die Systemmatrix $M_{i,j}$ aus (2.20) berücksichtigt allerdings noch keine Wirbelbelegungen. Im Falle einer konstanten Wirbelbelegung γ auf allen Panelen erweitert sich die Dimension von $M_{i,j}$ von $n \times n$ auf $(n+1) \times (n+1)$. Dadurch müssen folgende zusätzliche Elemente bestimmt werden:

$$M_{i,n} = \sum_{j=0}^{n-1} A_{i,j}^{(t)}, \quad i = 0, 1, \ldots, n-1, \tag{2.28}$$

$$M_{n,j} = A_{0,j}^{(t)} + A_{n-1,j}^{(t)}, \quad j = 0, 1, \ldots, n-1, \tag{2.29}$$

$$M_{n,n} = -\sum_{j=0}^{n-1} \left[A_{0,j}^{(n)} + A_{n-1,j}^{(n)} \right], \tag{2.30}$$

und

$$b_n = -V_\infty [\cos(\alpha - \theta 0) + \cos(\alpha - \theta_{n-1})]. \tag{2.31}$$

Gleichung (2.26) erweitert sich damit zu

$$v_i^{(t)} = \sum_{j=0}^{n-1} A_{i,j}^{(t)} q_j - \gamma \sum_{j=0}^{n-1} A_{i,j}^{(n)} + V_\infty \cos(\alpha - \theta_i). \tag{2.32}$$

[7] [4]

3 Bericht der Arbeit

Im Folgenden werden die wesentlichen Erkenntnisse der Arbeit präsentiert.

3.1 Vorbereitung der Daten

Die Daten der Profile wurden der UIUC Airfoil Data Site[1] entnommen. Die Daten lagen dabei überwiegend im Selig- oder Lednicer-Format vor.[2]
Im Lednicer-Format wird in der ersten Zeile die Bezeichnung des Profils angegeben. In der zweiten Zeile wird die Anzahl der Koordinatenpaare der Ober- und Unterseite angegeben. Ab der dritten Zeile werden die Koordinaten der Panelenden (x_i, y_i) von $x = 0$ bis $x = 1$ für die Oberseite des Profils angegeben. Danach folgt eine Leerzeile. Danach werden die Koordinaten der Panelenden (x_i, y_i) von $x = 0$ bis $x = 1$ für die Unterseite des Profils angegeben.
Im Selig-Format wird in der ersten Zeile die Bezeichnung des Profils angegeben. Ab der zweiten Zeile werden die Koordinaten der Panelenden (x_i, y_i) im Gegenuhrzeigersinn, beginnend bei $x = 1$ angegeben.

Es wurde eine Routine implementiert, welche Daten im Lednicer-Format in das Selig-Format überführt. Dies wurde bewerkstelligt, indem die Koordinatenpaare der Oberseite invertiert wurden. Anschließend wurden aus den Daten die Header-Zeilen entfernt.
Unter Verwendung der Methode .write_panels() wurde für die Profile eine .csv-Datei mit den Werten X_i, Y_i, θ_i, l_i pro Panel erstellt (eine Beispieldatei ist in Anhang A: Lednicer- und Selig-Format gezeigt. Ebenso wurden Graphen erzeugt, welche sowohl die Geometrie des Profils als auch die Neigungswinkel und Längen der einzelnen Panele darstellen. Zur einfacheren Betrachtung sind die Graphen der Neigungswinkel und Panellängen in die beiden Seiten des Profils aufgeteilt.
Ein solcher Graph ist in Abbildung 3.1 an einem NACA-0012-Profil gezeigt. Die Symmetrie des Profils spiegelt sich in der gewählten Darstellung der Neigungswinkel und Panellängen wieder.

[1] `https://m-selig.ae.illinois.edu/ads/coord_database.html`

[2] Eine Gegenüberstellung der beiden Formate ist in Anhang A: Lednicer- und Selig-Format zu finden.

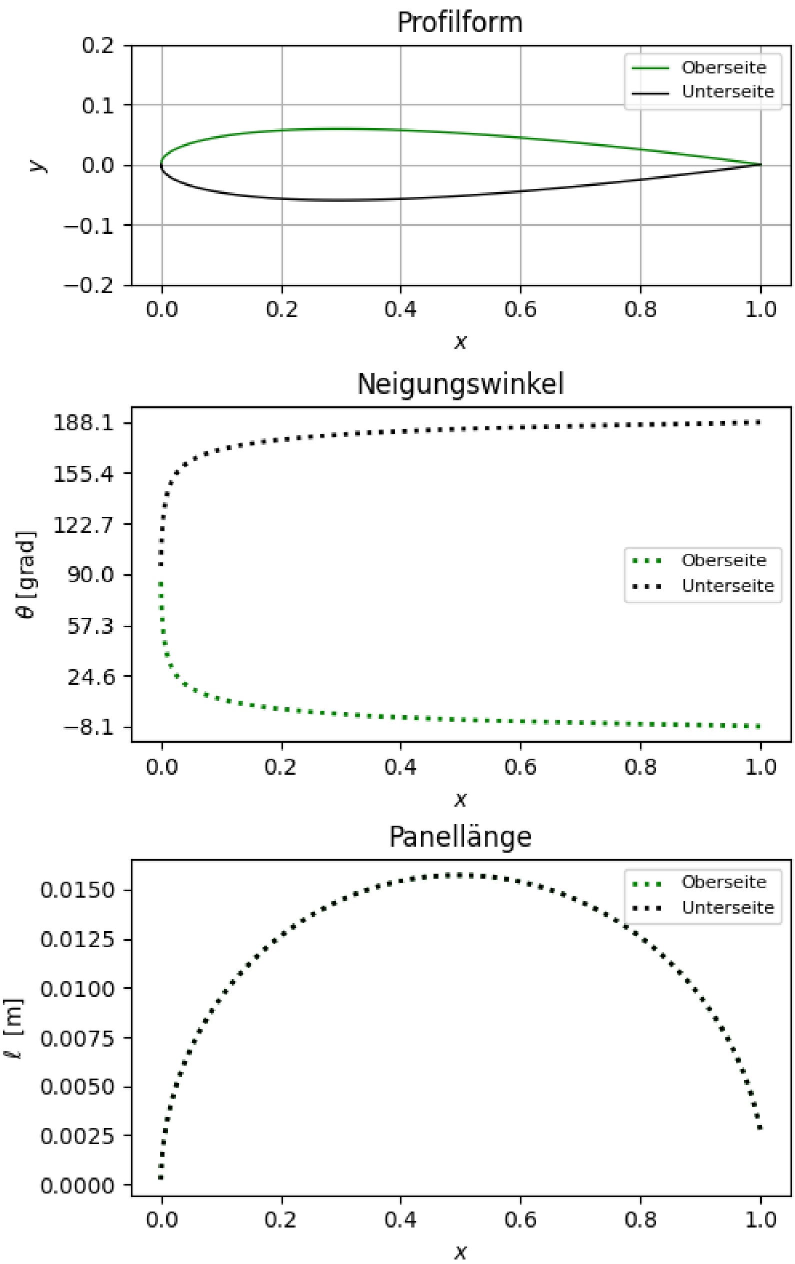


Abbildung 3.1: NACA-0012-Profil. Aufgrund der Symmetrie des Profils sind die Panellängen für Ober- und Unterseite deckungsgleich

X_i	Y_i	θ_i	l_i
0.93	-0.18	-112.5	0.38
0.68	-0.43	-157.5	0.38
0.32	-0.43	157.5	0.38
0.07	-0.18	112.5	0.38
0.07	0.18	67.5	0.38
0.32	0.43	22.5	0.38
0.68	0.43	-22.5	0.38
0.93	0.18	-67.5	0.38

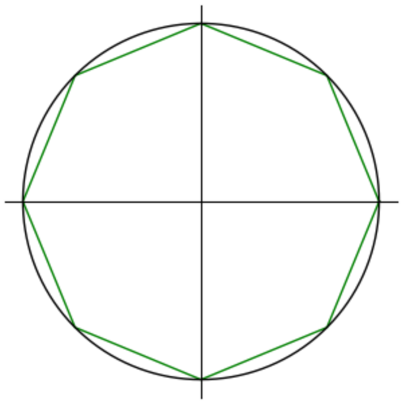


Abbildung 3.2: Achtseitiger Zylinder als Approximation eines Kreiszylinders sowie zugehörige Panelparameter

3.2 Untersuchungen am Kreiszylinder

Als nächstes wurden Untersuchungen an Kreiszylindern vorgenommen. Zur Generierung der Koordinatenpaare eines Kreiszylinders mit n gleich langen Seiten wurde die Funktion make_cylinder verwendet (siehe Anhang C: Codeausschnitte).
Die Panelparameter sowie die Systemmatrix und sämtliche damit verbundene Werte wurden entsprechend den Gleichungen (2.1) bis (2.3) sowie (2.20) bis (2.32) berechnet.

3.2.1 Achtseitiger Kreiszylinder

Berechnung der Systemmatrix

Für den in Abbildung 3.2 gezeigten Kreisylinder wurde die Systemmatrix, gerundet auf zwei Dezimalstellen, zu folgenden Werten berechnet:

$$
M_{i,j} = \begin{pmatrix}
0.5 & 0.056 & 0.064 & 0.065 & 0.065 & 0.065 & 0.064 & 0.056 \\
0.056 & 0.5 & 0.056 & 0.064 & 0.065 & 0.065 & 0.065 & 0.064 \\
0.064 & 0.056 & 0.5 & 0.056 & 0.064 & 0.065 & 0.065 & 0.065 \\
0.065 & 0.064 & 0.056 & 0.5 & 0.056 & 0.064 & 0.065 & 0.065 \\
0.065 & 0.065 & 0.064 & 0.056 & 0.5 & 0.056 & 0.064 & 0.065 \\
0.065 & 0.065 & 0.065 & 0.064 & 0.056 & 0.5 & 0.056 & 0.064 \\
0.064 & 0.065 & 0.065 & 0.065 & 0.064 & 0.056 & 0.5 & 0.056 \\
0.056 & 0.064 & 0.065 & 0.065 & 0.065 & 0.064 & 0.056 & 0.5
\end{pmatrix}
$$

Lösung des wirbelfreien Gleichungssystems

Im Folgenden wurde die Lösung des linearen Gleichungssystems mit der obigen Systemmatrix und den Inhomogenitäten

$$
b_i = -V_\infty \sin(\alpha - \theta_i), \quad i = 0, 1, \ldots, n-1, \tag{3.1}
$$

berechnet. Daraus wurde der Vektor der Quellenbelegung zunächst mit V_∞ und α als beliebigen Variablen berechnet:

$$
\vec{q} = \begin{pmatrix}
-0.16V_\infty \sin(\alpha - 5.89) - 0.17V_\infty \sin(\alpha - 5.11) - 0.17V_\infty \sin(\alpha - 4.32) - 0.17V_\infty \sin(\alpha - 3.53) - 0.16V_\infty \sin(\alpha - 2.75) - 0.12V_\infty \sin(\alpha - 1.96) + 2.13V_\infty \sin(\alpha - 1.18) - 0.12V_\infty \sin(\alpha - 0.39) \\
-0.12V_\infty \sin(\alpha - 5.89) - 0.16V_\infty \sin(\alpha - 5.11) - 0.17V_\infty \sin(\alpha - 4.32) - 0.17V_\infty \sin(\alpha - 3.53) - 0.17V_\infty \sin(\alpha - 2.75) - 0.16V_\infty \sin(\alpha - 1.96) - 0.12V_\infty \sin(\alpha - 1.18) + 2.13V_\infty \sin(\alpha - 0.39) \\
2.13V_\infty \sin(\alpha - 5.89) - 0.12V_\infty \sin(\alpha - 5.11) - 0.16V_\infty \sin(\alpha - 4.32) - 0.17V_\infty \sin(\alpha - 3.53) - 0.17V_\infty \sin(\alpha - 2.75) - 0.17V_\infty \sin(\alpha - 1.96) - 0.16V_\infty \sin(\alpha - 1.18) - 0.12V_\infty \sin(\alpha - 0.39) \\
-0.12V_\infty \sin(\alpha - 5.89) + 2.13V_\infty \sin(\alpha - 5.11) - 0.12V_\infty \sin(\alpha - 4.32) - 0.16V_\infty \sin(\alpha - 3.53) - 0.17V_\infty \sin(\alpha - 2.75) - 0.17V_\infty \sin(\alpha - 1.96) - 0.17V_\infty \sin(\alpha - 1.18) - 0.16V_\infty \sin(\alpha - 0.39) \\
-0.16V_\infty \sin(\alpha - 5.89) - 0.12V_\infty \sin(\alpha - 5.11) + 2.13V_\infty \sin(\alpha - 4.32) - 0.12V_\infty \sin(\alpha - 3.53) - 0.16V_\infty \sin(\alpha - 2.75) - 0.17V_\infty \sin(\alpha - 1.96) - 0.17V_\infty \sin(\alpha - 1.18) - 0.17V_\infty \sin(\alpha - 0.39) \\
-0.17V_\infty \sin(\alpha - 5.89) - 0.16V_\infty \sin(\alpha - 5.11) - 0.12V_\infty \sin(\alpha - 4.32) + 2.13V_\infty \sin(\alpha - 3.53) - 0.12V_\infty \sin(\alpha - 2.75) - 0.16V_\infty \sin(\alpha - 1.96) - 0.17V_\infty \sin(\alpha - 1.18) - 0.17V_\infty \sin(\alpha - 0.39) \\
-0.17V_\infty \sin(\alpha - 5.89) - 0.17V_\infty \sin(\alpha - 5.11) - 0.16V_\infty \sin(\alpha - 4.32) - 0.12V_\infty \sin(\alpha - 3.53) + 2.13V_\infty \sin(\alpha - 2.75) - 0.12V_\infty \sin(\alpha - 1.96) - 0.16V_\infty \sin(\alpha - 1.18) - 0.17V_\infty \sin(\alpha - 0.39) \\
-0.17V_\infty \sin(\alpha - 5.89) - 0.17V_\infty \sin(\alpha - 5.11) - 0.17V_\infty \sin(\alpha - 4.32) - 0.16V_\infty \sin(\alpha - 3.53) - 0.12V_\infty \sin(\alpha - 2.75) + 2.13V_\infty \sin(\alpha - 1.96) - 0.12V_\infty \sin(\alpha - 1.18) - 0.16V_\infty \sin(\alpha - 0.39)
\end{pmatrix}.
$$

Nach Einsetzen der Werte $V_\infty = 1$ und $\alpha = 0$ vereinfacht sich der Vektor zu einem leichter zu handhabenden Wert. Die Ergebnisse für Quellenbelegung, Tangentialgeschwindigkeiten und Druckbeiwerte sind in Tabelle 3.1 zusammengefasst. Ebenfalls von Interesse ist die Größe $\sum_{i=0}^{n-1} q_i l_i$. Diese Summe sollte für einen geschlossenen Körper null ergeben, da der Körper sonst durch den Fluss seine Masse verändern würde (vgl. [3]). Tatsächlich ergibt sich:

$$
\sum_{i=0}^{n-1} q_i l_i \approx 4.44 \cdot 10^{-16}.
$$

Dieser sehr kleine Fehler ergibt sich vermutlich aus Rundungsfehlern, die bei der Berechnung der Eckpunkte des Kreiszylinders entstehen.

Tabelle 3.1: Ergebnisse für den achtseitigen Zylinder mit $V_\infty = 1, \alpha = 0$

i	q_i	$v_i^{(t)}$	c_{p_i}
0	-2.19	-0.77	0.41
1	-0.91	-1.85	-2.41
2	0.91	-1.85	-2.41
3	2.19	-0.77	0.41
4	2.19	0.77	0.41
5	0.91	1.85	-2.41
6	-0.91	1.85	-2.41
7	-2.19	0.77	0.41

3.2.2 Untersuchungen an allgemeinen Kreiszylindern

Die Potentialtheorie liefert ein exaktes Resultat für den Druckbeiwert auf der Oberfläche eines Zylinders: [4]

$$c_p^{\text{exakt}}(\varphi) = 2\cos(2\varphi) - 1. \tag{3.2}$$

Dabei bezeichnet φ den Winkel der Flächennormale des Kreiszylinders welche ins Innere des Profils zeigt. Mit unserer Definition des Neigungswinkels θ gilt

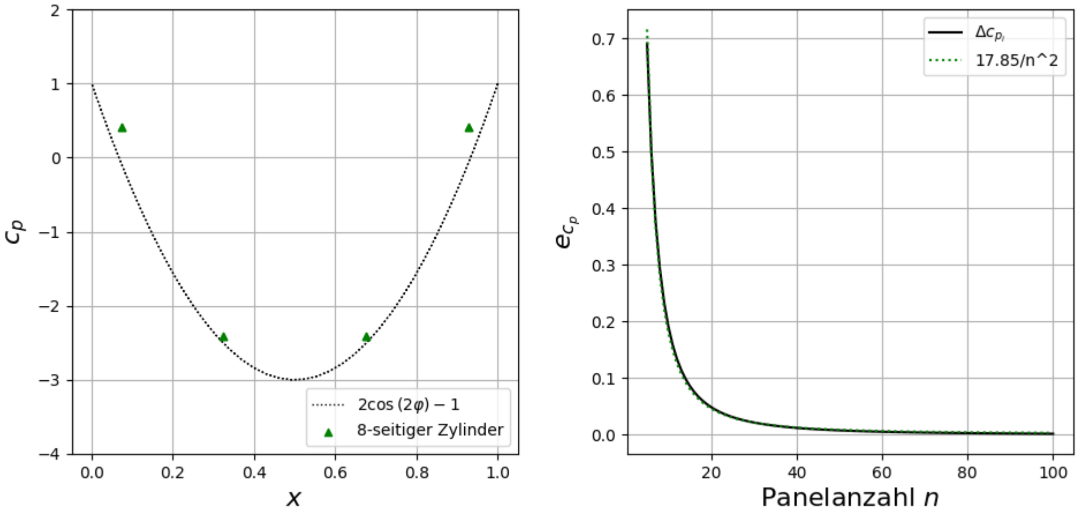
$$\varphi = \theta + \frac{\pi}{2}. \tag{3.3}$$

Abbildung 3.3a zeigt die Abweichung der Druckbeiwerte eines achtseitigen Zylinders vom theoretischen Resultat. Es ergibt sich eine Abweichung dadurch, dass die Mittelpunkte des achtseitigen Zylinders nicht auf der Kreisfläche liegen. Der Fehler ergibt sich zu:

$$\begin{aligned} c_p^{\text{exakt}}(\varphi) &= 2\cos(2\varphi) - 1 & (3.4)\\ &= 1 - 4\sin^2\varphi & (3.5)\\ &= 1 - 4\left(\frac{Y}{R}\right)^2. & (3.6) \end{aligned}$$

Der Fehler, der bei der Approximation entsteht, ergibt sich folglich zu

$$\begin{aligned} \Delta c_{p_i} &= |c_{p_i} - c_p^{\text{exakt}}(\varphi)| & (3.7)\\ &= |c_{p_i} - [1 - 4(Y_i/R)^2]\,|. & (3.8) \end{aligned}$$



(a) (b)

Abbildung 3.3: a) Abweichung der Druckbeiwerte eines achtseitigen Zylinders vom theoretisch ermittelten Wert b) Graph der Abweichung des mittleren Fehlers vom exaken Wert.

Variation der Panelanzahl

Das Programm wurde nun für Kreiszylinder zwischen 5 und 100 Panelen durchlaufen und die mittlere absolute Abweichung e_{c_p} für jeden Kreiszylinder festgehalten (Abbildung 3.3b). Aus den experimentellen Daten wurde mittels Kurvenanpassung für den absoluten Fehler in Abhängigkeit der Panelanzahl n die Beziehung

$$e_{c_p}(n) \approx \frac{17.85301}{n^2} + 0.00183,$$

ermittelt.

Rotation des achtseitigen Zylinders

Als nächstes wurden die Werte γ, c_a, sowie die Druckbeiwerte für verschiedene Winkel im Bereich $\alpha \in [-15°, 15°]$ bestimmt. Die Ergebnisse sind in Tabelle 3.2 aufgetragen.

Tabelle 3.2: Ergebnisse für den achtseitigen Zylinder mit $V_\infty = 1, \alpha \in [-15°, 15°]$

α	γ	c_a	c_{p_0}	c_{p_1}	c_{p_2}	c_{p_3}	c_{p_4}	c_{p_5}	c_{p_6}	c_{p_7}
-15	0.51	-2.93	0.45	-3.26	-5.06	-1.88	0.95	-0.23	-1.26	0.45
-14	0.48	-2.74	0.45	-3.22	-4.88	-1.68	0.98	-0.35	-1.34	0.45
-13	0.44	-2.55	0.44	-3.18	-4.7	-1.49	0.99	-0.47	-1.42	0.44
-12	0.41	-2.35	0.44	-3.13	-4.53	-1.3	1.0	-0.6	-1.5	0.44
-11	0.38	-2.16	0.44	-3.08	-4.35	-1.12	1.0	-0.73	-1.58	0.44
-10	0.34	-1.96	0.43	-3.03	-4.17	-0.95	0.99	-0.87	-1.66	0.43
-9	0.31	-1.77	0.43	-2.98	-3.99	-0.78	0.97	-1.01	-1.74	0.43
-8	0.28	-1.57	0.43	-2.92	-3.81	-0.62	0.94	-1.15	-1.82	0.43
-7	0.24	-1.38	0.42	-2.86	-3.63	-0.46	0.9	-1.3	-1.9	0.42
-6	0.21	-1.18	0.42	-2.81	-3.46	-0.32	0.86	-1.45	-1.97	0.42
-5	0.17	-0.99	0.42	-2.74	-3.28	-0.18	0.81	-1.6	-2.05	0.42
-4	0.14	-0.79	0.42	-2.68	-3.1	-0.04	0.74	-1.76	-2.12	0.42
-3	0.1	-0.59	0.42	-2.62	-2.93	0.08	0.67	-1.92	-2.2	0.42
-2	0.07	-0.39	0.41	-2.55	-2.76	0.2	0.6	-2.08	-2.27	0.41
-1	0.03	-0.2	0.41	-2.48	-2.58	0.31	0.51	-2.25	-2.34	0.41
0	-0.0	0.0	0.41	-2.41	-2.41	0.41	0.41	-2.41	-2.41	0.41
1	-0.03	0.2	0.41	-2.34	-2.25	0.51	0.31	-2.58	-2.48	0.41
2	-0.07	0.39	0.41	-2.27	-2.08	0.6	0.2	-2.76	-2.55	0.41
3	-0.1	0.59	0.42	-2.2	-1.92	0.67	0.08	-2.93	-2.62	0.42
4	-0.14	0.79	0.42	-2.12	-1.76	0.74	-0.04	-3.1	-2.68	0.42
5	-0.17	0.99	0.42	-2.05	-1.6	0.81	-0.18	-3.28	-2.74	0.42
6	-0.21	1.18	0.42	-1.97	-1.45	0.86	-0.32	-3.46	-2.81	0.42
7	-0.24	1.38	0.42	-1.9	-1.3	0.9	-0.46	-3.63	-2.86	0.42
8	-0.28	1.57	0.43	-1.82	-1.15	0.94	-0.62	-3.81	-2.92	0.43
9	-0.31	1.77	0.43	-1.74	-1.01	0.97	-0.78	-3.99	-2.98	0.43
10	-0.34	1.96	0.43	-1.66	-0.87	0.99	-0.95	-4.17	-3.03	0.43
11	-0.38	2.16	0.44	-1.58	-0.73	1.0	-1.12	-4.35	-3.08	0.44
12	-0.41	2.35	0.44	-1.5	-0.6	1.0	-1.3	-4.53	-3.13	0.44
13	-0.44	2.55	0.44	-1.42	-0.47	0.99	-1.49	-4.7	-3.18	0.44
14	-0.48	2.74	0.45	-1.34	-0.35	0.98	-1.68	-4.88	-3.22	0.45
15	-0.51	2.93	0.45	-1.26	-0.23	0.95	-1.88	-5.06	-3.26	0.45

3.3 Auswertung ausgewählter Profile

Im Folgenden wurde das entwickelte Programm auf die drei Profile NACA0012, NASA SC(2)-0614 und Taylor Zone-40 angewandt. Für jedes Profil wurden die Werte γ und c_a unter verschiedenen Anströmwinkeln α berechnet.
An dieser Stelle wurde das Programm um die Möglichkeit erweitert, den Geschwindigkeitsvektor an jedem Punkt des Profils zu berechnen. Dies spiegelt sich in (2.24) und (2.25) wieder, für welche dann

$$\xi_j = (x - X_j)\cos\theta_j + (y - Y_j)\sin\theta_j, \tag{3.9}$$

und

$$\eta_j = -(x - X_j)\sin\theta_j + (y - Y_j)\cos\theta_j, \tag{3.10}$$

gelten. (2.21) und (2.27) vereinfachen sich ebenfalls zu

$$A_j^{(n)} = \sin(\theta_j)I_j + \cos(\theta_j)J_j; \tag{3.11}$$

$$A_j^{(t)} = \cos(\theta_j)I_j - \sin(\theta_j)J_j. \tag{3.12}$$

Der Geschwindigkeitsvektor im Punkt $\vec{x} = (x, y)$ ergibt sich also zu:

$$\vec{v}(x,y) = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} \sum_{j=0}^{n-1} A_j^{(t)} q_j - \gamma \sum_{j=0}^{n-1} A_j^{(n)} + V_\infty \cos(\alpha) \\ \sum_{j=0}^{n-1} A_j^{(n)} q_j + \gamma \sum_{j=0}^{n-1} A_j^{(t)} + V_\infty \cos(\alpha) \end{pmatrix}. \tag{3.13}$$

Diese Geschwindigkeitsvektoren wurden für die oben genannten Profile für ein Gitter der Dimension 200×200 und den Punkten

$$\vec{x} = (x,y) = \{-0.5, -0.5 + n\Delta x, \ldots, 1,5\} \times \{-0.2, -0.2 + n\Delta y, \ldots, 0.2\};$$
$$\Delta x = \frac{1.5 - (-0.5)}{200}, \quad \Delta y = \frac{0.2 - (-0.2)}{200}, \quad n \in \mathbb{N} \cap (0, 200),$$

berechnet und sind in Form eines Strömungsgraphens dargestellt. Auch wurde ein Konturplot für die sich ergebenden Druckbeiwerte im Umgebungsfeld des Profils generiert.
Ebenfalls wurde in Form einer Colormap für die Panele die Werte für Quellbelegung und Druckbeiwerte auf der Profiloberfläche ermittelt und dargestellt. Für sämtliche oben genannten Graphen wurde für die Anströmgeschwindigkeit $V_\infty = 1$, für den Anströmwinkel $\alpha = 5°$ gewählt.

3.3.1 NACA0012-Profil

α	γ	c_a
-11	0.32	-1.32
-7	0.21	-0.84
-3	0.09	-0.36
1	-0.03	0.12
5	-0.15	0.6
9	-0.27	1.08
13	-0.38	1.55
17	-0.5	2.02

Tabelle 3.3: Ergebnisse des NACA0012-Profils

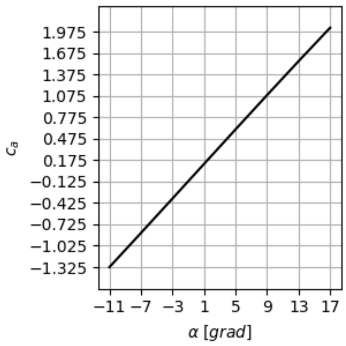
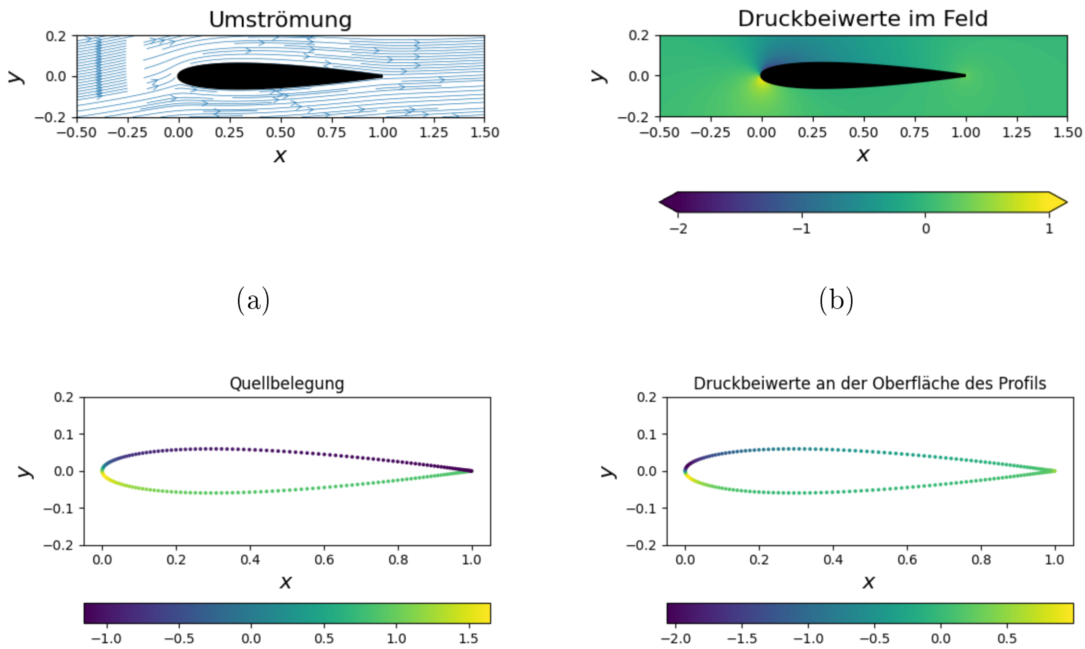


Abbildung 3.4: Graph des Auftriebsbeiwerts



(a) (b) (c) (d)

Abbildung 3.5: a) Umströmung und b) Druckbeiwerte im Feld c) Quellbelegung und d) Druckbeiwerte pro Panel $\mathcal{C}_i$ unter dem Anströmwinkel $\alpha = 5°$

3.3.2 NASA SC(2)-0614-Profil

α	γ	c_a
-11	0.3	-1.26
-7	0.19	-0.77
-3	0.07	-0.27
1	-0.05	0.22
5	-0.17	0.71
9	-0.29	1.2
13	-0.41	1.69
17	-0.53	2.16

Tabelle 3.4: Ergebnisse des NASA SC(2)-0614-Profils

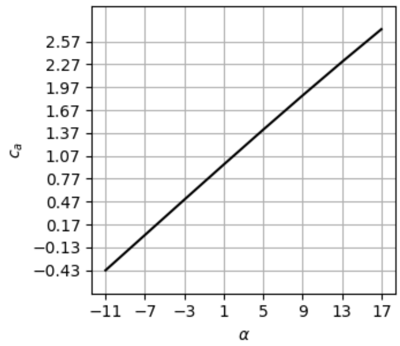
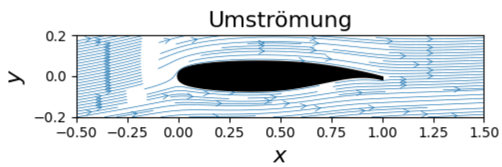
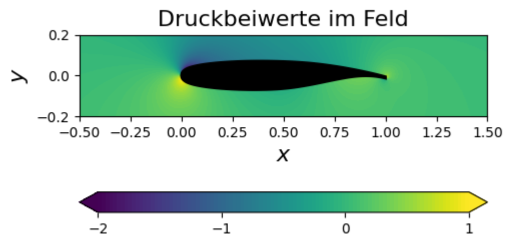


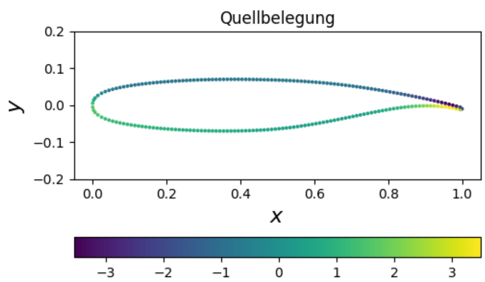
Abbildung 3.6: Graph des Auftriebsbeiwerts



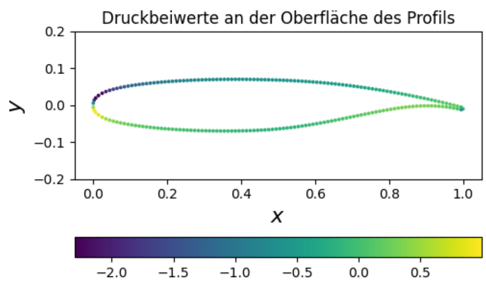
(a)



(b)



(c)



(d)

Abbildung 3.7: a) Umströmung und b) Druckbeiwerte im Feld c) Quellbelegung und d) Druckbeiwerte pro Panel $\mathcal{C}_i$ unter dem Anströmwinkel $\alpha = 5°$

3.3.3 Taylor Zone-40-Profil

α	γ	c_a
-11	0.31	-1.23
-7	0.19	-0.78
-3	0.08	-0.32
1	-0.04	0.14
5	-0.15	0.59
9	-0.27	1.05
13	-0.38	1.49
17	-0.49	1.94

Tabelle 3.5: Ergebnisse des Taylor Zone-40-Profils

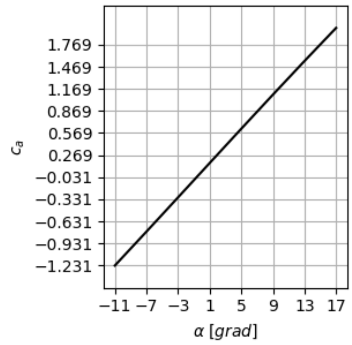
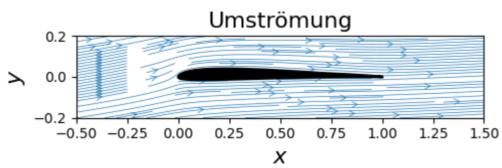
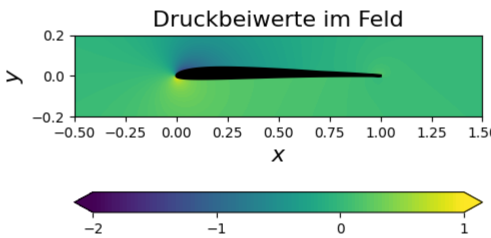


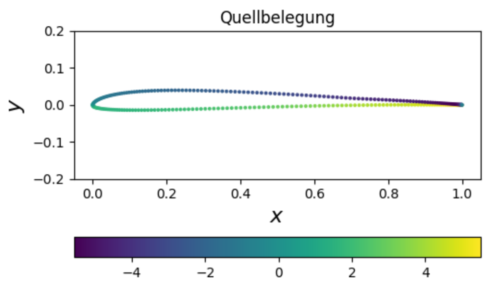
Abbildung 3.8: Graph des Auftriebsbeiwerts



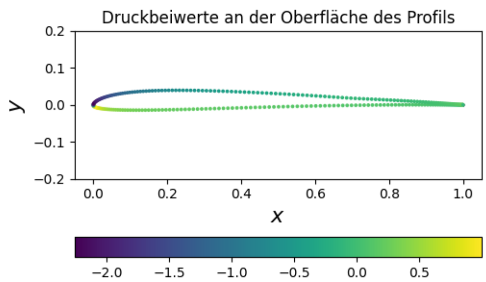
(a)



(b)

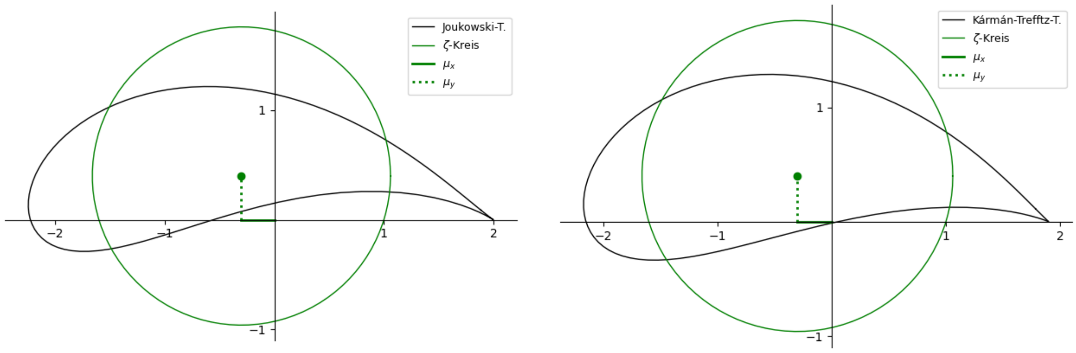


(c)



(d)

Abbildung 3.9: a) Umströmung und b) Druckbeiwerte im Feld c) Quellbelegung und d) Druckbeiwerte pro Panel $\mathcal{C}_i$ unter dem Anströmwinkel $\alpha = 5°$



(a) (b)

Abbildung 3.10: Vergleich der a) Joukowski-Transformation mit der b) Kármán-Trefftz-Transformation. Für beide Transformationen wurde $\mu_x = 0.3, \mu_y = 0.4$ gewählt; für die Kármán-Trefftz-Transformation wurde $n = 1.9$ gesetzt.

3.4 Abschätzung der Fehlerordnung anhand des Joukowski-Profils

Die Joukowski-Transformation ist eine konforme Abbildung, deren Transformation als Ergebnis Tragflächenprofile liefert. Die Transformationsgleichung lautet:

$$z = \zeta + \frac{a^2}{\zeta}, \tag{3.14}$$

mit einem reellen Parameter a welcher im Folgenden $a = 1$ gesetzt wird. Dabei ist $z = x + iy$ eine komplexe Zahl im Bildraum und $\zeta = \chi + i\eta$ eine komplexe Zahl im Ursprungsraum.
Ein Joukowski-Profil wird im z-Raum durch die Anwendung der Joukowski-Transformation auf einen Kreis im ζ-Raum erzeugt. Einsetzen von z und ζ in (3.14) ergibt die realen und imaginären Komponenten

$$x = \frac{\chi\,(\chi^2 + \eta^2 + 1)}{\chi^2 + \eta^2}, \quad y = \frac{\eta\,(\chi^2 + \eta^2 - 1)}{\chi^2 + \eta^2}. \tag{3.15}$$

Durch Verschiebung des Mittelpunkts des Kreises in der ζ-Ebene können so verschieden Profile generiert werden. Die Abbildung hat allerdings an den Stellen

$z = \pm a = \pm 1$ eine Singularität. Deshalb legt man einen der Punkte (hier $z = -1$) ins Innere des Kreises und den Punkt $z = 1$ auf den Kreis. In diesem Fall ergibt sich an der Hinterkante ein Winkel von 0°.

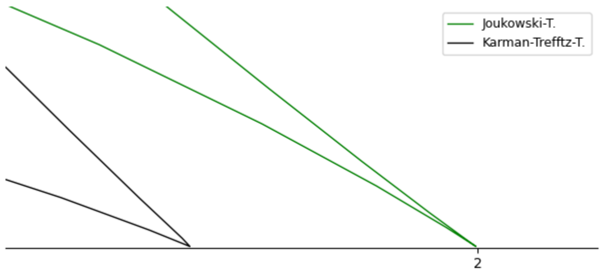


Abbildung 3.11: Vergleich der Hinterkantenwinkel des Joukowski- und Kármán-Trefftz-Profils

Ein Profil mit einem Winkel ungleich Null an der Hinterkante lässt sich durch eine verwandte Transformation, die Kármán-Trefftz-Transformation

$$z = nb\frac{(\zeta + b)^n + (\zeta - b)^n}{(\zeta + b)^n - (\zeta - b)^n}, \tag{3.16}$$

erzeugen, wiederum mit einem reellen Parameter b (im Folgenden $b = 1$) und $n \in [1, 2]$, wobei sich für $n = 1$ der Kreis in der ζ-Ebene und für $n = 2$ die Joukowski-Transformation ergibt. Die beiden Transformationen sind in Abbildung 3.10 unter gleichen Parametern dargestellt. Abbildung 3.11 gibt einen näheren Blick auf die Profilhinterkante.
Die analytische Lösung einer ebenen Potentialströmung um einen Kreiszylinder ist bekannt (siehe 3.2.2). Damit ist auch eine Lösung für Joukowski-Profile gegeben. Für die Zirkulation $\Gamma = \sum_{i=1}^{n-1} \gamma l_i$ ergibt sich

$$\Gamma = 4V_\infty R \sin\left(\alpha + \arcsin\frac{\mu_y}{R}\right). \tag{3.17}$$

Mit dem Kutta-Jukowski-Theorem für den dynamischen Auftrieb $F_a = \rho\Gamma V_\infty$ ist über den dynamischen Auftrieb mit $c_a = \frac{F_a}{pt} = \frac{2F_a}{\rho V_\infty}$ ein Ausdruck für den Auftriebsbeiwert gegeben:

$$c_a = \frac{8RV_\infty \sin\left(\alpha + \arcsin\frac{\mu_y}{R}\right)}{V_\infty t}. \tag{3.18}$$

Es wurde nun für Joukowski-Profile mit den Parametern $\mu_x = 0.2, \mu_y = 0.1$ für 10 bis 200 Panele der Auftriebsbeiwert berechnet und mit dem theoretischen Resultat aus (3.18) verglichen. Es ist ersichtlich, dass sich der Auftriebsbeiwert ab einer

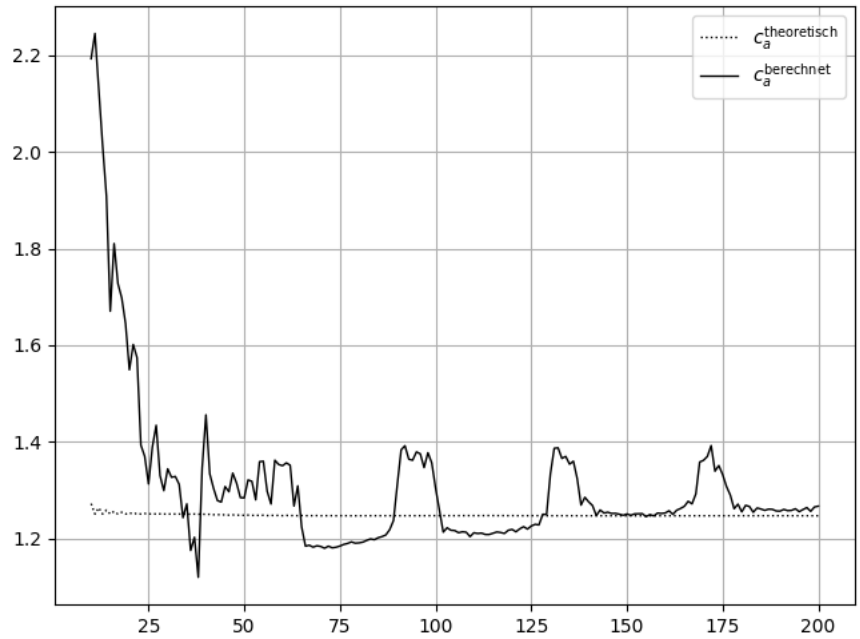


Abbildung 3.12: Vergleich des theoretisch ermittelten Werts für den Auftriebsbeiwert eines Joukowski-Profils mit $\mu_x = 0.2, \mu_y = 0.1$ und dem berechneten Wert.

Panelanzahl von ungefähr 75 stabil eingeschwungen hat, bis auf Schwankungen, welche vermutlich auf der Diskretisierungsfunktion des Joukowski-Profils beruhen. Für den mittleren relativen Fehler eines Joukowski-Profils mit mindestens 75 Panelen wurde der Wert

$$\begin{aligned}\frac{\Delta c_a}{c_a^{\text{theoretisch}}} &= \frac{|c_a^{\text{theoretisch}} - c_a^{\text{exakt}}|}{c_a^{\text{theoretisch}}} \\ &= 0.033 = 3.3\%,\end{aligned}$$

ermittelt. [1] [3]

Literatur

[1] Carlos Abello, Pedro Pablo Cárdenas Alzate und Jose Granada. „A survey of Joukowski airfoil and von Karman vortex street“. In: *Contemporary Engineering Sciences* 11 (Jan. 2018), S. 4921–4928. DOI: 10.12988/ces.2018.810540.

[2] J.J. Alonso. *Hess-Smith Panel Method*. 2005. URL: http://aero-comlab.stanford.edu/aa200b/lect_notes/lect3-4.pdf (besucht am 28. 07. 2022).

[3] Lorena Barba und Olivier Mesnard. „Aero Python: classical aerodynamics of potential flow using Python“. In: *Journal of Open Source Education* 2.16 (2019), S. 45. DOI: 10.21105/jose.00045. URL: https://doi.org/10.21105/jose.00045.

[4] T. Cebeci. *An Engineering Approach to the Calculation of Aerodynamic Flows*. 1. Aufl. Berlin Heidelberg: Springer, 1999. ISBN: 978-3-540-66181-8.

[5] Sean Gillies u. a. *Shapely: manipulation and analysis of geometric objects*. toblerity.org, 2007–. URL: https://github.com/Toblerity/Shapely.

[6] Charles R. Harris u. a. „Array programming with NumPy“. In: *Nature* 585.7825 (Sep. 2020), S. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[7] J.L. Hess und A.M.O. Smith. „Calculation of Potential Flow About Arbitrary Bodies“. In: *Progress in Aerospace Sciences* 8 (1966), S. 117–137.

[8] J. D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.

[9] Aaron Meurer u. a. „SymPy: symbolic computing in Python“. In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[10] Guido Van Rossum und Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[11] Pauli Virtanen u. a. „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python“. In: *Nature Methods* 17 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2.

Anhang A: Lednicer- und Selig-Format

Es folgt eine Gegenüberstellung des Selig-Formats (links) mit dem Lednicer-Format (rechts) am Beispiel des NACA M13-Profils.

```
NACA M13 AIRFOIL
1.00000000 0.00440000
0.95000000 0.01056000
0.90000000 0.01642000
0.80000000 0.02924000
0.70000000 0.04186000
0.60000000 0.05378000
0.50000000 0.06360000
0.40000000 0.06942000
0.30000000 0.07014000
0.20000000 0.06276000
0.15000000 0.05542000
0.10000000 0.04468000
0.07500000 0.03791000
0.05000000 0.02964000
0.02500000 0.01937000
0.01250000 0.01278000
0.00000000 0.00000000
0.00000000 0.00000000
0.01250000 -0.00812000
0.02500000 -0.00813000
0.05000000 -0.00791000
0.07500000 -0.00439000
0.10000000 -0.00222000
0.15000000 0.00192000
0.20000000 0.00506000
0.30000000 0.00804000
0.40000000 0.00772000
0.50000000 0.00550000
0.60000000 0.00228000
0.70000000 -0.00064000
0.80000000 -0.00226000
0.90000000 -0.00228000
0.95000000 -0.00144000
1.00000000 0.00000000
```

```
NACA M13 AIRFOIL
17 17
0.000000 0.000000
0.012500 0.012780
0.025000 0.019370
0.050000 0.029640
0.075000 0.037910
0.100000 0.044680
0.150000 0.055420
0.200000 0.062760
0.300000 0.070140
0.400000 0.069420
0.500000 0.063600
0.600000 0.053780
0.700000 0.041860
0.800000 0.029240
0.900000 0.016420
0.950000 0.010560
1.000000 0.004400

0.000000 0.000000
0.012500 -0.008120
0.025000 -0.008130
0.050000 -0.007910
0.075000 -0.004390
0.100000 -0.002220
0.150000 0.001920
0.200000 0.005060
0.300000 0.008040
0.400000 0.007720
0.500000 0.005500
0.600000 0.002280
0.700000 -0.000640
0.800000 -0.002260
0.900000 -0.002280
0.950000 -0.001440
1.000000 0.000000
```

Anhang B: Beispielausgabe der Methode .write_panels()

Hier ist die Ausgabe der ersten 30 Zeilen der AirfoilProfile-Methode .write_panels() gezeigt, am Beispiel des NASA: HSNLF(1)-0213 Profils.

```
X_i,Y_i,theta_i,l_i
1.0,0.0,-90.0,0.0013398
0.9950043,-0.0009728,-176.53028176,0.01000975
0.9850132,-0.0016043,-176.23647604,0.01001239
0.97752025,-0.0021092,-175.96223567,0.00500753
0.96253695,-0.00330425,-175.33539764,0.02505448
0.937568,-0.00552425,-174.50339006,0.02508173
0.92258845,-0.0069828,-174.11527646,0.00501915
0.910107,-0.0083076,-173.89772702,0.020084
0.8876414,-0.0107718,-173.61456462,0.02511682
0.8626817,-0.01365475,-173.20816903,0.02513479
0.8377253,-0.01677495,-172.53929414,0.02516746
0.82275295,-0.01875585,-172.08379489,0.00503831
0.81027805,-0.02055885,-171.69908973,0.02017082
0.79032025,-0.0235841,-171.06256939,0.02020137
0.77784795,-0.02556705,-170.5814527,0.00505667
0.76288615,-0.0283095,-169.42015294,0.02536633
0.73795785,-0.03345505,-167.26194135,0.02555034
0.7130314,-0.0387352,-168.82200575,0.0254135
0.6905846,-0.0425474,-172.30336929,0.02014367
0.6781074,-0.04417745,-173.57353486,0.00502377
0.6631284,-0.0456803,-174.41027033,0.02508508
0.63815875,-0.0478474,-175.6702679,0.02504498
0.61318205,-0.0495071,-176.72682089,0.02502072
0.58819985,-0.05077395,-177.46737688,0.02500893
0.56321345,-0.05173975,-178.10560524,0.02500196
0.5382233,-0.05243635,-178.70102613,0.02499842
0.5132296,-0.0528728,-179.29814725,0.02499728
0.4882332,-0.0531112,-179.608979,0.02499798
0.4632347,-0.05320265,-179.97181003,0.0249996
0.43823365,-0.0531076,179.53618988,0.02500332
```

Anhang C: Codeausschnitte

Hier sind ausgewählte Codeausschnitte gezeigt, welche zur Lösung der Problemstellungen geschrieben wurden.

Die Klasse Panel

Die Klasse Panel modelliert die Panels $\mathcal{C}_i$ eines gegebenen Profils. Gespeichert werden neben den charakteristischen Parametern X_i, Y_i, θ_i, l_i auch die Quellbelegung q_i, die Tangentialgeschwindigkeit $v_i^{(t)}$ unter gegebenen Anströmwinkel α und -geschwindigkeit V_∞, und der resultierende Druckbeiwert c_{p_i}.

```
class Panel:

    def __init__(self, xa, ya, xb, yb):
        self.xa, self.ya = xa, ya
        self.xb, self.yb = xb, yb

        self.xm = (xa + xb) / 2
        self.ym = (ya + yb) / 2
        self.length = np.sqrt((xb - xa) ** 2 + (yb - ya) ** 2)
        self.theta = np.arctan2(yb - ya, xb - xa)

        self.q = None
        self.vt = None
        self.cp = None
```

Die Klasse AirfoilProfile

Die Klasse AirfoilProfile modelliert ein gegebenes Profil. Sie speichert neben wählbaren Namen und der ihr zugewiesenen Panele auch die Profiltiefe t und sämtliche Systemparameter ξ_{ij}, η_{ij}, I_{ij}, J_{ij}, $A_{ij}^{(n)}$, $A_{ij}^{(t)}$, M_{ij}.
Durch einen Aufruf der Methode .solve(V, a) mit gegebenen α und V_∞ werden für alle zugeordneten Panele die Quellstärken, Tangentialgeschwindigkeiten und Druckbeiwerte berechnet (und in den jeweiligen Klassenvariablen abgespeichert). Dabei wird ebenfalls die Genauigkeit der Approximation $\sum q_i l_i$ berechnet. Die Methode kann mit dem Schlüsselwortargument vortex=True aufgerufen werden, wodurch ebenfalls der Auftriebsbeiwert c_a, sowie die Wirbelbewegung γ
berechnet werden. Die Methode .write_panels() erzeugt eine .csv-Datei, welche für jedes Panel die Werte $X_i, Y_i \theta_i, l_i$ in eine Zeile schreibt (siehe Anhang A: Lednicer- und Selig-Format).

Die Methode .compute_free_vt(x, y, V, a) ermöglicht die Berechnung der Tangentialgeschwindigkeiten an jedem Punkt (x_i, y_i) des Profils. Diese werden als Tupel von der Methode zurückgegeben.

```
class AirfoilProfile:
    def __init__(self, panels, name=None, vortex=True):
        self.panels = panels
        self.name = name
        self.x = [panel.xa for panel in self.panels]
        self.y = [panel.ya for panel in self.panels]
        self.len = len(self.panels)
        self.vortex = vortex
        self.shape = Polygon([(a,b) for a,b in zip(self.x, self.y)])

        self.U = sum([panel.length for panel in self.panels])
        self.t = abs(max(self.x) - min(self.x))

        self.xi = self.__xi()
        self.eta = self.__eta()
        self.I = self.__i()
        self.J = self.__j()
        self.An = self.__an()
        self.At = self.__at()
        self.M = self.__m()

        self.solve_state = None
        self.ca = None
        self.accuracy = None
        self.gamma = None

    def write_panels(self, filename, n=2):
        header = ["X_i", "Y_i", "theta_i", "l_i"]
        with open(filename, "w+", encoding='UTF8',newline="") as file:
            writer = csv.writer(file)
            writer.writerow(header)
            for panel in self.panels:
                writer.writerow([round(panel.xm,n), round(panel.ym,n),
                    round(panel.theta*180/np.pi,n), round(panel.length,n)])

    def __xi(self, x=None, y=None):
        panels = self.panels
```

```
        n_panels = len(panels)
        if x is not None and y is not None:
            n = 1
        else:
            n = len(panels)
        Xi = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                if x is not None and y is not None:
                    i_xm, i_ym = x, y
                else:
                    i_xm, i_ym = panels[i].xm, panels[i].ym
                pj = panels[j]
                Xi[i][j] = (i_xm - pj.xm) * np.cos(pj.theta) + (i_ym - pj.
                    ym) * np.sin(pj.theta)
        return Xi

    def __eta(self, x=None, y=None):
        panels = self.panels
        n_panels = len(panels)
        if x is not None and y is not None:
            n = 1
        else:
            n = len(panels)
        Eta = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                if x is not None and y is not None:
                    i_xm, i_ym = x, y
                else:
                    i_xm, i_ym = panels[i].xm, panels[i].ym
                pj = panels[j]
                Eta[i][j] = - (i_xm - pj.xm) * np.sin(pj.theta) + (i_ym -
                    pj.ym) * np.cos(pj.theta)
        return Eta

    def __i(self, Xi=None, Eta=None):
        panels = self.panels
        if Xi is None and Eta is None:
            Xi = self.xi
```

```python
            Eta = self.eta
        n_panels = len(panels)
        n = Xi.shape[0]
        II = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                pj = panels[j]
                if i == j and n > 1:
                    II[i][j] = 0
                else:
                    II[i][j] = (1 / (4 * np.pi)) * np.log(((pj.length + 2 * Xi[
                        i][j]) ** 2 + 4 * (Eta[i][j] ** 2)) / ((pj.length - 2
                        * Xi[i][j]) ** 2 + 4 * (Eta[i][j] ** 2)))
        return II

    def __j(self, Xi=None, Eta=None):
        panels = self.panels
        if Xi is None and Eta is None:
            Xi = self.xi
            Eta = self.eta
        n_panels = len(panels)
        n = Xi.shape[0]
        JJ = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                pj = panels[j]
                if i == j and n > 1:
                    JJ[i][j] = 0.5
                else:
                    JJ[i][j] = (1 / (2 * np.pi)) * np.arctan((pj.length - 2 *
                        Xi[i][j])/ (2 * Eta[i][j])) + (1 / (2 * np.pi)) * np.
                        arctan((pj.length + 2 * Xi[i][j])/ (2 * Eta[i][j]))
        return JJ

    def __an(self, II=None, JJ=None):
        panels = self.panels
        if II is None and JJ is None:
            II = self.I
            JJ = self.J
        n_panels = len(panels)
```

```
        n = II.shape[0]
        if n == 1:
            i_theta = 0
        AN = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                if n > 1:
                    i_theta = panels[i].theta
                pj = panels[j]
                AN[i][j] = - np.sin(i_theta - pj.theta) * II[i][j] + np.cos(
                    i_theta - pj.theta) * JJ[i][j]
        return AN

    def __at(self, II=None, JJ=None):
        panels = self.panels
        if II is None and JJ is None:
            II = self.I
            JJ = self.J
        n_panels = len(panels)
        n = II.shape[0]
        if n == 1:
            i_theta = 0
        AT = np.empty((n, n_panels), dtype=float)
        for i in range(n):
            for j in range(n_panels):
                if n > 1:
                    i_theta = panels[i].theta
                pj = panels[j]
                AT[i][j] = np.cos(i_theta - pj.theta) * II[i][j] + np.sin(
                    i_theta - pj.theta) * JJ[i][j]
        return AT

    def __m(self):
        panels = self.panels
        AN = self.An
        AT = self.At
        vortex = self.vortex
        n = len(panels)
        if vortex:
            MM = np.empty((n + 1, n + 1), dtype=float)
```

```python
        else:
            MM = np.empty((n, n), dtype=float)
        if vortex:
            MM[:-1, :-1] = AN
            MM[:-1, -1] = np.sum(AT, axis=1)

            r = np.empty(n + 1, dtype=float)
            r[:-1] = AT[0, :] + AT[n - 1, :]
            r[-1] = -np.sum(AN[0, :] + AN[n - 1, :])
            MM[-1, :] = r
        else:
            MM = AN
        return MM

    def solve(self, V=1, a=5):
        a = np.radians(a)
        b = self.__b(V, a)

        self.__q(b) # setzt auch self.gamma
        self.__vt(V, a)
        self.__cp(V)
        self.__ca(V)
        self.__accuracy()
        self.solve_state = (V,a)

    def compute_free_vt(self, x, y, V=1, a=5):
        if (V,a) != self.solve_state:
            self.solve(V=V, a=a)
        a = np.radians(a)
        point = Point(x,y)
        if point.within(self.shape) or self.shape.touches(point):
            vtx = np.nan
            vty = np.nan
        else:
            xi = self.__xi(x=x, y=y)
            eta = self.__eta(x=x, y=y)
            I = self.__i(Xi=xi, Eta=eta)
            J = self.__j(Xi=xi, Eta=eta)
            An = self.__an(II=I, JJ=J)
            At = self.__at(II=I, JJ=J)
```

```
            vtx = sum([At[0][j] * self.panels[j].q for j in range(self.len)])
                - self.gamma * sum(
                [An[0][j] for j in range(self.len)]) + V * np.cos(a)

            vty= sum([An[0][j] * self.panels[j].q for j in range(self.len)]) +
                 self.gamma * sum(
                [At[0][j] for j in range(self.len)]) + V * np.sin(a)

        return vtx, vty

    def __b(self, V, a):
        panels = self.panels
        vortex = self.vortex
        n = len(panels)
        if vortex:
            B = np.empty(n + 1, dtype=float)
        else:
            B = np.empty(n, dtype=float)
        for i in range(n):
            pi = panels[i]
            B[i] = -V * np.sin(a - pi.theta)
        if vortex:
            B[-1] = -V * (np.cos(a - panels[0].theta) + np.cos(a - panels[-1].
                theta))
        return B

    def __q(self, B):
        panels = self.panels
        qs = np.linalg.solve(self.M, B)

        for i, panel in enumerate(panels):
            panel.q = qs[i]
        if self.vortex:
            self.gamma = qs[-1]

    def __vt(self, V, a):
        panels = self.panels
        AN = self.An
        AT = self.At
```

```
    n = len(panels)
    if self.vortex:
        VT = np.empty(n, dtype=float)
        for i in range(n):
            VT[i] = sum([AT[i][j] * panels[j].q for j in range(n)]) \
                    - self.gamma * sum([AN[i][j] for j in range(n)]) \
                    + V * np.cos(a - panels[i].theta)
    else:
        VT = np.empty(n, dtype=float)
        for i in range(n):
            VT[i] = sum([AT[i][j] * panels[j].q for j in range(n)]) \
                    + V * np.cos(a - panels[i].theta)

    for i, panel in enumerate(panels):
        panel.vt = VT[i]

def __cp(self, V):
    for panel in self.panels:
        panel.cp = 1 - (panel.vt / V) ** 2

def __ca(self, V):
    self.ca = 2 / (V * self.t) * sum([panel.vt * panel.length for panel in
        self.panels])

def __accuracy(self):
    self.accuracy = sum([panel.q * panel.length for panel in self.panels])
```

make_cylinder(r,n)

Diese Funktion wurde verwendet, um die x- und y-Koordinaten eines Kreiszylinder mit Radius r und n gleich langen Panels zu generieren. Es wurde dabei besonderes Augenmerk darauf gelegt, dass es an den Endpunkten durch Rundungsfehler nicht zu einem disjunkten Körper kommt.

```
def cylinder(r=1, n=8):
    a = np.linspace(0, 360, num=n+1, endpoint=True) / 180 * np.pi

    x = r * np.cos(a)
    y = r * np.sin(a)
    if abs(x[0] - x[-1]) <= 10 ** (-15):
        x[-1] = x[0]
```

```
    if abs(y[0] - y[-1]) <= 10 ** (-15):
        y[-1] = y[0]

    return x, y
```

make_panels(x,y, reverse=False)

```
    if type(x) is not np.ndarray:
        x = np.array(x)
    if type(y) is not np.ndarray:
        y = np.array(y)
    if (x[0], y[0]) != (x[-1], y[-1]):
        x = np.append(x, x[0])
        y = np.append(y, y[0])
    if reverse:
        x = np.flipud(x)
        y = np.flipud(y)
    n = len(x) - 1
    panels = np.array([Panel(x[i], y[i], x[i + 1], y[i + 1]) for i in range(n)
        ])
    return panels
```

parsecoords(filename)

```
    x, y = np.loadtxt(filename, dtype=float, unpack=True)
    return x, y
```

Joukowsky-Profil-Generator

```
def joukowski_transfrom(zeta):
    z = zeta + 1 / zeta
    return z
def make_joukowski(mux=0.2, muy=0.1, N=100):
    center = -mux + muy * 1j
    R = np.sqrt((1 + mux) ** 2 + muy ** 2)

    theta = np.linspace(0, 2 * np.pi, N)
    Xc = np.real(center) + R * np.cos(theta)
    Yc = np.imag(center) + R * np.sin(theta)
```

```
    p = joukowski_transfrom(Xc + Yc * 1j)
    Xp, Yp = np.real(p), np.imag(p)
    return Xp, Yp, R, muy
```

Kármán-Trefftz-Profil-Generator

```
def karman_trefftz_transform(zeta, n):
    z = n * ((zeta + 1) ** n + (zeta - 1) ** n) / ((zeta + 1) ** n - (zeta -
        1) ** n)
    return z
def make_karman_trefftz(mux=0.2, muy=0.1, n=1.9, N=100):
    center = -mux + muy * 1j
    R = np.sqrt((1 + mux) ** 2 + muy ** 2)

    theta = np.linspace(0, 2 * np.pi, N)
    Xc = np.real(center) + R * np.cos(theta)
    Yc = np.imag(center) + R * np.sin(theta)
    tx = np.real(center)
    ty = np.imag(center)

    p = karman_trefftz_transform(Xc + Yc * 1j, n)
    Xp, Yp = np.real(p), np.imag(p)
    return Xp, Yp, R, muy
```