Entwicklung eines Programms zur Auswertung von Strömungssimulationen für Hubschrauberkonfigurationen

DIPLOMARBEIT

für die Prüfung zum

Diplom-Ingenieur (Berufsakademie)

der Fachrichtung Informationstechnik

an der Berufsakademie Mannheim

von

ENRICO TAPPERT

September 2005

Bearbeitungszeitraum	3 Monate
Kurs	TIT02BNS
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt e. V. in der Helmholtz-Gemeinschaft 38108 Braunschweig
Gutachter der Ausbildungsfirma	Thorsten Schwarz
Gutachter der Studienakademie	Dipl.-Ing. Joachim Kabamba

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich meine Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Braunschweig, den 16. September 2005

Declaration of Academic Honesty

I hereby declare to have written this Diploma Thesis
on my own, having used only the listed resources and tools.

Braunschweig, 16. September 2005

Kurzfassung

Das Institut für Aerodynamik und Strömungstechnik des Deutschen Zentrums für Luft- und Raumfahrt entwickelt numerische und experimentelle Strömungssimulations-Verfahren namens FLOWer und TAU. Diese ermöglichen es, die Aerodynamik von Hubschraubern, Flug- und Raumfahrzeugen schon vor der experimentellen Erprobung zu simulieren. Damit gelangen von vornherein gute Entwürfe zur physikalischen Analyse in den Windkanal beziehungsweise zum Flugversuch. Die Strömungssimulations-Verfahren lösen die Reynolds-gemittelten Navier-Stokes-Gleichungen zur Berechnung von reibungsbehafteten Strömungen. Hinsichtlich der Auswertung der numerischen Simulation werden beispielsweise globale Kräfte wie der Auftrieb oder der Gesamtwiderstand des Körpers benötigt.

Im Rahmen dieser Diplomarbeit entstand ein Programm, welches die Ergebnisse der Strömungssimulations-Verfahren auswerten kann. Hierfür werden Dateien mit strukturiertem Datenformat von FLOWer und mit unstrukturiertem Format von TAU eingelesen. Auf dieser Basis können Profile erstellt werden, die zum Beipiel die Daten eines Hubschrauberrumpfes beinhalten. Daraufhin können verschiedene Berechnungen durchgeführt werden, die zum Beispiel den Auftrieb und den Gesamtwiderstand für das jeweilige Profil ermitteln. Des Weiteren können Schnitte durch das Profil berechnet werden, um die Daten an diesen einzelnen Stellen auszuwerten.
Der Benutzer kann aus einem großen Umfang die benötigten Funktionalitäten in einer Skriptdatei in einer bestimmten Reihenfolge festlegen, um das Programm anschließend automatisiert ablaufen zu lassen. Die Definition der Ablaufreihenfolge in einer Skriptdatei erfolgt im Hinblick dessen, da somit die Integration in die Arbeitsumgebung des Instituts für Aerodynamik und Strömungstechnik gewährleistet ist.

Abstract

Abstract englisch

Inhaltsverzeichnis

Abkürzungsverzeichnis

Analython	**Analy**se, Py**thon**
API	**A**pplication **P**rogramming **I**nterface, Programmierschnittstelle
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt e.V. in der Helmholtz-Gemeinschaft
GUI	**G**raphical **U**ser **I**nterface, grafische Benutzerschnittstelle
IDE	**I**ntegrated **D**evelopment **E**nvironment, Entwicklungsumgebung
Numpy	**Num**erical **Py**thon
OMG	**O**bject **M**anagement **G**roup
UML	**U**nified **M**odelling **L**anguage, Modellierungssprache

Notation

a	Skalierungsfaktor
ϱ_∞	Dichte der freien Anströmung
μ	Reibung der freien Anströmung
$\vec{B}$	Bezugspunktvektor
c_f	Reibungsbeiwert (coefficient of friction)
c_{f_∞}	Reibungsbeiwert der freien Anströmung
c_p	Druckbeiwert (coefficient of pressure)
d	Abstand
D	Widerstand (drag)
$\vec{E}$	Ortsvektor eines Punktes in einer Ebene
$\vec{F}$	Gesamtkraftvektor
$\vec{F_f}$	Reibungskraftvektor
$\vec{F_p}$	Druckkraftvektor
$\vec{G}$	Ortsvektor eines Punktes auf einer Geraden
$\vec{H}$	Hebelarmvektor
L	Auftrieb (lift)
$\vec{M}$	Momentvektor
$\vec{n}$	Normalenvektor
$\vec{P}, \vec{Q}, \vec{R}, \vec{T}$	Ortsvektoren verschiedener Punkte
p	Druck (pressure)
p_∞	Druck der freien Anströmung
$\vec{q}$	Geschwindigkeitsvektor, $\vec{q} = (u, v, w)^T$
q_∞	Geschwindigkeitsbetrag der freien Anströmung
t	Tangente
$\vec{s}$	Oberflächenvektor (surface)
V_∞	Anströmgeschwindigkeit/Fluggeschwindigkeit

1 Einleitung

Das Institut für Aerodynamik und Strömungstechnik des Deutschen Zentrums für Luft- und Raumfahrt (DLR) entwickelt numerische und experimentelle Strömungssimulations-Verfahren. Diese ermöglichen es, die Aerodynamik von Hubschraubern, Flug- und Raumfahrzeugen schon vor der experimentellen Erprobung zu simulieren. Damit gelangen von vornherein gute Entwürfe zur physikalischen Analyse in den Windkanal beziehungsweise zum Flugversuch. Die numerische Strömungssimulation ist somit ein unverzichtbares Werkzeug für die Entwicklung von Luft- und Raumfahrzeugen, da sie die Entwicklungszeit und die Entwicklungskosten eines Luft- oder Raumfahrzeugs wesentlich verringert.

Für die numerischen Strömungssimulationen werden im Institut das strukturierte Rechenverfahren FLOWer und das unstruktierte Verfahren TAU eingesetzt. Beide lösen die Reynolds-gemittelten Navier-Stokes-Gleichungen zur Berechnung von reibungsbehafteten Strömungen. Die Strömungsgrößen im Feld beziehungsweise die Drücke und Reibungskräfte auf den Körperoberflächen des betrachteten Luft- oder Raumfahrzeugs sind die resultierenden Ergebnisse.

Hinsichtlich der Auswertung der numerischen Simulation werden beispielsweise globale Kräfte wie der Auftrieb oder der Gesamtwiderstand des Körpers benötigt. Diese globalen Größen müssen aus den lokalen Größen der gesamten Körperoberfläche oder einzelner Oberflächenkomponenten bestimmt werden. Dies erfordert den Einsatz von Auswerteprogrammen, welche speziell für einen Konfigurationstyp, beispielsweise für Flugzeuge, angepasst sind.

Die vorliegende Diplomarbeit beschreibt die Entwicklung eines Auswerteprogramms für Hubschrauberkonfigurationen.

2 Aufgabenbeschreibung

Im Institut für Aerodynamik und Strömungstechnik wurde vor einigen Jahren ein Auswerteprogramm für Hubschrauberkonfigurationen entwickelt. Dieses kann die Ergebnisse des Strömungslösers FLOWer jedoch nicht die des ebenso eingesetzten TAU auswerten. Des Weiteren ist es in seinem Funktionsumfang sehr eingeschränkt. Infolge der ständigen Weiterentwicklung des Strömungslösers FLOWer hat sich dessen Ausgabe der Ergebnisse verändert. Dies hat zur Folge, dass das existierende Auswerteprogramm modifiziert werden müsste, um weiterhin korrekte Berechnungen durchzuführen. Aufgrund der fehlenden Modularität ist es allerdings nur schwer editier- und erweiterbar.

Aus diesen Gründen soll im Rahmen dieser Arbeit ein Auswerteprogramm für Hubschrauberkonfigurationen entwickelt werden. Die Teilaufgaben ergeben sich aus der Aufgabenstellung wie folgt:

- Analyse und Definition der Anforderungen an das Auswerteprogramm
- Erstellen eines Konzepts zur Realisierung des Auswerteprogramms
- Implementierung des Auswerteprogramms
  - Eingabe, Speicherung, Ausgabe der Daten
  - Berechnung von Kräften und Momenten
  - Berechnung von Schnittlinien sowie deren Kräfte und Momente
  - Skalierung einzelner Daten
- Nachweis der Funktionstüchtigkeit des Auswerteprogramms

3 Theoretische Grundlagen

Innerhalb dieser Diplomarbeit werden Berechnungen verwendet, welche auf unterschiedlichen Netzstrukturen basieren. Diese Strukturen werden in Bezug auf [4] in diesem Kapitel erklärt. Hinsichtlich der Verarbeitung von Daten werden darüber hinaus die Formate von relevanten Dateien beschrieben. Des Weiteren findet eine nähere Erläuterung der verwendeten Berechnungsvorschriften statt.

3.1 Netzaufbau

Im Institut für Aerodynamik und Strömungstechnik des DLR werden Flugzeuge, Hubschrauber und einzelne Teile davon, zum Beispiel ein Flügel, numerisch simuliert. Hierfür wurden Strömungssimulations-Verfahren entwickelt, die aufgrund eines Rechennetzes um den jeweiligen Körper, das so genannte Profil, die Strömung und damit zusamenhängende Größen berechnen. Die Berechnung findet innerhalb eines vorher bestimmten Gebietes statt, in dem sich das jeweilige Profil befindet. Dies ist in Abbildung 1 verdeutlicht. Um das Profil wird ein dreidimensionales Gebiet abgegrenzt, welches in viele kleinere Gebiete unterteilt wird. Damit entsteht im gesamten Gebiet ein Netz. Die Eckpunkte der kleinen Teilgebiete bilden die Netzpunkte. Das bedeutet, es wird eine Diskretisierung des Raumes um das Profil herum durchgeführt. Die sich daraus ergebenden Diskretisierungsstellen (Netzpunkte) werden zusammen als numerisches Netz bezeichnet. Die Netzpunkte sind zum Zwecke der grafischen Darstellung miteinander verbunden. Im Netz erfolgt an den Netzpunkten die Berechnung verschiedener Größen.

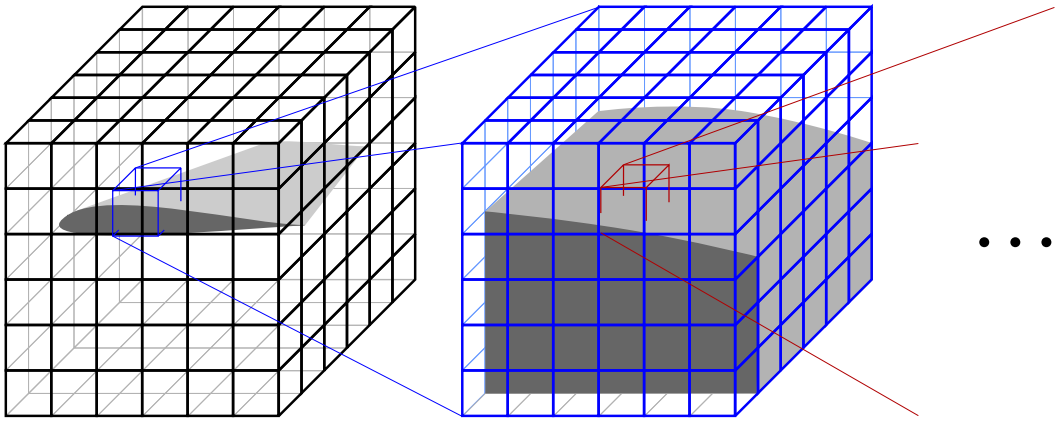


Abbildung 1: Flügelprofil mit umgebendem Netz

Das in Abbildung 1 dargestellte Gebiet bildet ein dreidimensionales Netz, dessen Netzpunkte im gesamten dreidimensionalen Strömungsgebiet liegen. Des Weiteren bilden die so genannten Oberflächennetze die Diskretisierungspunkte auf der dreidimensionalen Oberfläche des Profils ab. Außerdem existieren zweidimensionale Netze, deren Punkte in einer Ebene liegen.

In Bezug auf den Zusammenhang der Netzpunkte untereinander wird zwischen strukturierten und unstrukturierten Netzen unterschieden. Im Institut für Aerodynamik und Strömungstechnik wurden das Strömungssimulationsverfahren FLOWer auf der Basis von strukturierten Daten und das Verfahren TAU auf Basis von unstrukturierten Daten entwickelt, deren Ergebnisse die Grundlage für Berechnungen in dieser Diplomarbeit bilden. Strukturierte Daten werden dabei in strukturierten Dateien, unstrukturierte Daten werden in unstrukturierten Dateien gespeichert.

3.1.1 Strukturierte Netze

Sind die Punkte eines Netzes im Raum so angeordnet, dass durch Verbinden benachbarter Punkte ausschließlich gleichartige Polyeder entstehen, so spricht man von einem strukturierten Netz (vergleiche Abbildung 1 im vorherigen Kapitel). Je kleiner diese Polyeder sind, desto größer ist die Auflösung. Eine große Auflösung ermöglicht präzisere Ergebnisse bei den Berechnungen an den Netzpunkten.
Jeder Netzpunkt ist durch einen Indextripel (i, j, k) eindeutig bestimmt. Das kartesische Netz bildet das einfachste strukturierte Netz. Hierbei sind die Abstände der Diskretisierungspunkte voneinander konstant. Die Netzlinien stehen dabei senkrecht aufeinander.

Der Vorteil von strukturierten Netzen liegt in der einfachen Implementierbarkeit in Computerprogrammen. Die Netzpunkte und deren Strömungsgrößen können in mehrdimensionalen Feldern abgebildet werden. Eine Projektion des Indextripels (i, j, k) auf die Feldindizes ermöglicht den direkten Zugriff auf die Punkte.

3.1.2 Unstrukturierte Netze

In dreidimensionalen unstrukturierten Netzen sind die Diskretisierungspunkte beliebig im Raum verteilt. Ähnlich wie bei den strukturierten Netzen besitzt jeder Punkt eine Beziehung zu mehreren anderen Nachbarpunkten. Folglich sind verschiedene Polyederformen möglich, wobei innerhalb eines Netzes nur eine Form vorkommt, beispielsweise

Hexaeder. Der Zusammenhang zwischen den Nachbarpunkten muss vorgegeben und in Computerprogrammen explizit abgespeichert werden.

Der durch unstrukturierte Netze höhere Speicherplatzbedarf wirkt sich nachteilig gegenüber den strukturierten Netzen aus. Allerdings ist diese Vorgehensweise der Diskretisierung wesentlich flexibler bei der Gestaltung des Rechennetzes.

3.2 Dateiaufbau

Die beiden Rechenverfahren FLOWer und TAU beschreiben Werte für die Diskretisierungsstellen der Netze. Diese werden in verschiedenen Ausgabedateien gespeichert, von denen einige die Eingabedateien für das zu entwickelnde Auswerteprogramm darstellen. Dieses Kapitel beschreibt die Formate dieser Dateien. Hierbei werden nur die Aspekte des Formats betrachtet, die für diese Arbeit relevant sind.

3.2.1 Strukturierte Dateien

Die Ausgabedateien des strukturierten Strömungssimulationsverfahrens FLOWer beinhalten verschiedene Daten, die sich auf jeweils ein Profil beziehen. Ein Profil kann dabei aus mehreren so genannten Zonen bestehen.

Der Aufbau dieser Dateien folgt einem festgelegten Format. Dies hat den Vorteil, dass sie problemlos von anderen Programen weiterverarbeitet werden können. Eines dieser Programme ist Tecplot, mit dem die Daten der Dateien visualisiert werden können. Das Programm Tecplot wird in Kapitel 5.2.2 näher erläutert.
Die Abbildung 2 zeigt den prinzipiellen Aufbau des Beginns einer strukturierten als auch unstrukturierten Datei.

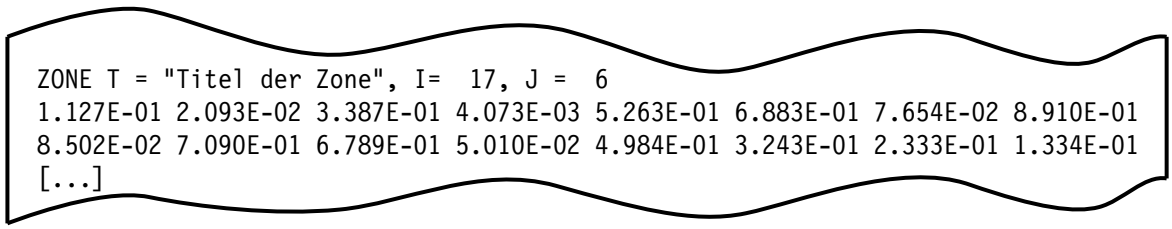
```
TITLE = "Titel des Profils"
VARIABLES = "x", "y", "z", "u", "v", "w", "cp", "cfinf"
# Kommentarzeile
```

Abbildung 2: Beginn einer Beispieldatei

In der ersten Zeile steht das Schlagwort TITLE, hinter dem der Titel für das Profil angegeben wird. In der nächsten Zeile folgt das Schlüsselwort VARIABLES. Dahinter

sind die Namen aller Variablen einer jeweiligen Datenzeile aufgeführt, deren genaue Daten im weiteren Verlauf folgen. Ein Raute-Symbol (#) am Beginn einer Zeile definiert diese als Kommentar, so dass sie bei weiteren Berechnungen nicht betrachtet werden.

Die folgenden Angaben können innerhalb einer Datei mehrmals auftreten, wobei der oben angesprochene Teil nur einmal vorkommt.
Ein Profil kann aus mehreren Zonen bestehen. Dies ist zum Beispiel dann der Fall, sobald ein Tragflügel und dessen Klappen getrennt voneinander in der Berechnung betrachtet werden. Hierfür steht in der Eingabedatei am Anfang jeder Zone das Schlüsselwort ZONE. Der dahinter angefügte Kennbuchstabe T zeigt an, dass darauf der Titel der Zone folgt. Bei den strukturierten Eingabedateien sind daraufhin die Kennbuchstaben I und J aufgeführt, welche, zusammen multipliziert, die Anzahl der Datenpunkte (Netzpunkte) der jeweiligen Zone angeben. Die Parameter I und J geben hierbei diese Anzahl wie in einer Matrix zeilen- und spaltenweise an. Dies wird in Abbildung 3 ersichtlich.
Nach der allgemeinen Zonenbeschreibung sind mehrere Zeilen mit verschiedenen Werten aufgeführt. Die Anzahl der Zeilen entspricht dem Produkt aus I und J. Daraus lässt sich schließen, dass eine Zeile die Werte der verschiedenen Variablen für einen Netzpunkt angibt. Dies bedeutet, dass eine Datenzeile so viele Werte aufzeigen muss, wie Variablen am Beginn der Datei aufgezählt sind. Durch den einer Matrix ähnlichen Aufbau kann nachvollzogen werden, welche Netzpunkte miteinander in Beziehung stehen.



```
ZONE T = "Titel der Zone", I=  17, J =  6
1.127E-01 2.093E-02 3.387E-01 4.073E-03 5.263E-01 6.883E-01 7.654E-02 8.910E-01
8.502E-02 7.090E-01 6.789E-01 5.010E-02 4.984E-01 3.243E-01 2.333E-01 1.334E-01
[...]
```

Abbildung 3: Zonenbeschreibung einer strukturierten Beispieldatei

3.2.2 Unstrukturierte Dateien

Die Ausgabedateien des unstrukturierten Rechenverfahrens TAU sind den strukturierten Dateien ähnlich. Der Beginn einer solchen Datei ist bei beiden identisch, mit entsprechend anderen Variablenbezeichnungen. Lediglich die Beschreibung der Zonen ist voneinander verschieden.

```
ZONE T = "Titel der Zone", N=  17, E =  6, ZONETYPE = "FEQuadrilateral"
1.127E-01 2.093E-02 3.387E-01 4.073E-03 5.263E-01 6.883E-01 7.654E-02 8.910E-01
8.502E-02 7.090E-01 6.789E-01 5.010E-02 4.984E-01 3.243E-01 2.333E-01 1.334E-01
[...]
12 7 3 15
5 9 11 17
[...]
```

Abbildung 4: Zonenbeschreibung einer unstrukturierten Beispieldatei

Wie in Abbildung 4 ersichtlich wird, sind das Schlüsselwort und der Kennbuchstabe T in der Zonenbeschreibung wie bei den strukturierten Dateien ebenso vorhanden. Anschließend folgen zwei andere Kennbuchstaben N und E. Jede Datenzeile unterhalb der Beschreibung stellt wiederum die Werte für einen Netzpunkt bereit. Die Anzahl der Datenzeilen entspricht dem Wert für den Kennbuchstaben N. N ist der Bezeichner für *number of **n**odes* (Anzahl der Knoten) in einem Netz.
Da allerdings die Strukturierung fehlt, muss explizit angegeben werden, welche Netzpunkte zu welchen anderen eine Verbindung besitzen. Dies geschieht nach den Datenzeilen. Es werden in jeweils einer Zeile die Nummern der Datenzeilen aufgeführt, die eine Nachbarschaft besitzen. Die Anzahl dieser Zeilen entspricht dem Wert für E. Der Kennbuchstabe steht für *number of **e**lements* (Anzahl der Verbindungselemente).
Das ebenso in der Zonenbeschreibung aufgeführte Schlagwort ZONETYPE gibt den Typ der jeweiligen Zone und damit den Aufbau des Netzes der jeweiligen Zone an. Dies ist zum Beispiel „FEQuadrilateral“ für einen viereckigen Aufbau und „FETriangle“ für einen dreieckigen Aufbau.

3.3 Mathematische Grundlagen

Im Rahmen dieser Arbeit ist es laut Aufgabenstellung (siehe Kapitel 2) notwendig einige Berechnungen durchzuführen. Diese finden auf den Oberflächennetzen eines jeweiligen Profils statt. Die verschiedenen Berechnungen und deren Vorschriften werden in diesem Kapitel näher erläutert.

3.3.1 Oberflächenvektoren

Die Oberflächenvektoren bilden die Grundlage der hier behandelten mathematische Berechnungen. Sie zeigen in jedem Netzpunkt in Richtung des Normalenvektors, sie stehen demnach senkrecht auf der Oberfläche. Der Betrag des Vektors gibt die Größe der ihn umgebendenden Oberfläche an. Die Abbildung 5 verdeutlicht diesen Sachverhalt für ein dreidimensionales strukturiertes Oberflächennetz. Die Vektoren $\vec{P}, \vec{Q}, \vec{R}, \vec{T}$ sind Ortsvektoren einzelner Netzpunkte. Mit A werden die jeweiligen zu berechnenden Flächen bezeichnet. Anhand der Abbildung 5 soll die Berechnung des Oberflächenvektors $\vec{s}$ im Punkt P beschrieben werden.

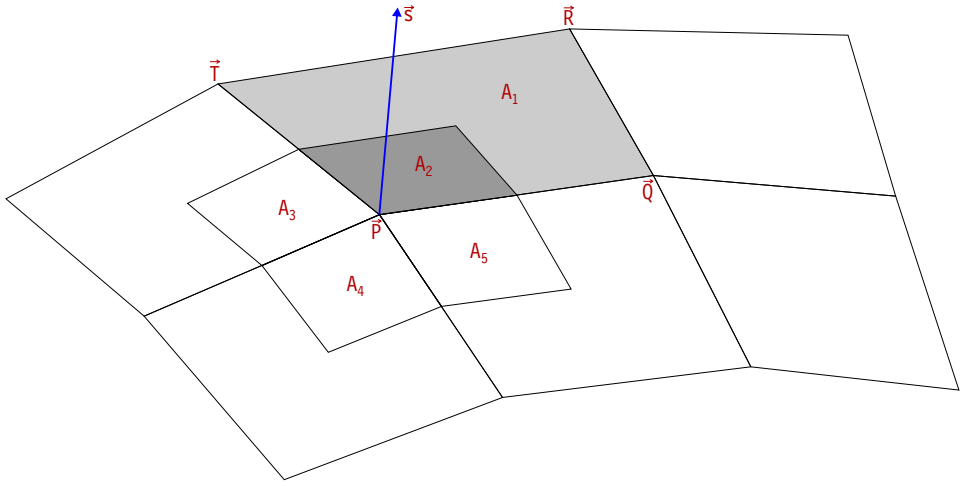


Abbildung 5: Netzausschnitt mit Oberflächenvektoren

Für jeden Netzpunkt ist der Ortsvektor gegeben. Am Beipielpunkt P gilt

$$\vec{P} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{1}$$

Mitels des Vektorproduktes lässt sich für jede Fläche die Größe bestimmen:

$$\vec{A_1} = \frac{1}{2} \cdot ((\vec{T} - \vec{Q}) \times (\vec{R} - \vec{P})) = \frac{1}{2} \cdot \left(\begin{pmatrix} Tx - Qx \\ Ty - Qy \\ Tz - Qz \end{pmatrix} \times \begin{pmatrix} Rx - Px \\ Ry - Py \\ Rz - Pz \end{pmatrix} \right) \tag{2}$$

mit $(\vec{T} - \vec{Q})$ und $(\vec{R} - \vec{P})$ als Richtungsvektoren. Die Fläche muss in diesem Beispiel mit $\frac{1}{4}$ dividiert werden, da zu jedem Netzpunkt ein Viertel aller angrenzenden Flächen zählt beziehungsweise jede Fläche vier Begrenzungspunkte besitzt, auf die sie aufgeteilt wird.

Durch Addition der resultierenden Flächenvektoren erhält man den Oberflächenvektor im Punkt P.

$$\vec{A_2} = \frac{1}{4} \cdot \vec{A_1} \tag{3}$$

$$\vec{s} = \vec{A_2} + \vec{A_3} + \vec{A_4} + \vec{A_5} \tag{4}$$

Grenzen an einen Netzpunkt weniger als vier Flächen, so werden diese ebenfalls mit einem Viertel zum Oberflächenvektor aufaddiert. Hierbei ist wiederum davon auszugehen, dass jede Fläche vier Begrenzungspunkte besitzt. Diese Fläche wird demnach auf alle vier Punkte gleich verteilt.

Wie in der Aufgabenstellung in Kapitel 2 beschrieben ist, soll das Auswerteprogramm Linien berechnen, die sich durch das Schneiden einer Ebene mit dem Oberflächennetz ergeben. Da eine Linie nur eine Ausdehnung in der Länge, aber keine in der Breite besitzt, müssen für diesen Fall die Oberflächenvektoren neu berechnet werden. Der Oberflächenvektor würde sonst die falsche Größe der angrenzenden Fläche aufzeigen und dies führt zu falschen Ergebnissen bei der Berechnung von Kräften und Momenten für diese Linie. Zu dem Zweck der Ermittlung der Werte für die Oberflächenvektoren werden für die angrenzenden Strecken (vom Bezugs- zu den Nachbarpunkten) die Längen berechnet. Jeweils die Hälfte dieser Längen wird zum neuen Oberflächenvektor hinzuaddiert. Hierbei wird folglich von einer Breite der Linie von *1* ausgegangen, da einerseits eine Berechnung mit der Breite *0* die Größe *0* der Fläche zur Folge hätte. Andererseits wird dabei auf Berechnungsvorschriften Bezug genommen, die beispielsweise im Strömungssimulationsverfahren TAU angewendet werden.

3.3.2 Beiwerte

beiwerte Der Begriff Beiwert stammt laut [11] aus älterer technischer Literatur und definiert einen Koeffizienten, also einen multiplikativen Faktor zu einem bestimmten Objekt.
Im weiteren Verlauf dieser Diplomarbeit sollen verschiedene Kräfte sowie der Auftrieb und Widerstand eines Profils berechnet werden. Dazu ist es notwendig, den Druckbeiwert c_p und den Reibungsbeiwert c_f zu kennen.

Dem Vorlesungsskript [10] entsprechend berechnet sich der Druckbeiwert an der Oberfläche eines Profils durch

$$c_p = \frac{p - p_\infty}{q_\infty} \tag{5}$$

mit

$$q_\infty = \frac{\varrho_\infty}{2} \cdot V_\infty^2 \tag{6}$$

Der Druckbeiwert ist im Rahmen dieser Arbeit stets gegeben, so dass hierauf nicht weiter eingegangen werden muss.

Der Reibungsbeiwert hingegen muss aus den vorhandenen Daten ermittelt werden. Er wird für jeden Netzpunkt berechnet. Grundlage hierfür bildet die Geschwindigkeit der Strömung um das Profil in diesen Punkten, welche als Geschwindigkeitsvektor $\vec{q}$ gegeben ist. Die Reibung entsteht durch die tangentiale Geschwindigkeit am jeweiligen Oberflächennetzpunkt, welche sich aus der Gesamtgeschwindigkeit ergibt.
Abbildung 6 zeigt eine zweidimensionale Ansicht eines Oberflächennetzes mit der Geschwindigkeit $\vec{q}$, der tangentialen Geschwindigkeit $\vec{q_t}$ und dem Normalenvektor $\vec{n}$ in Punkt P.

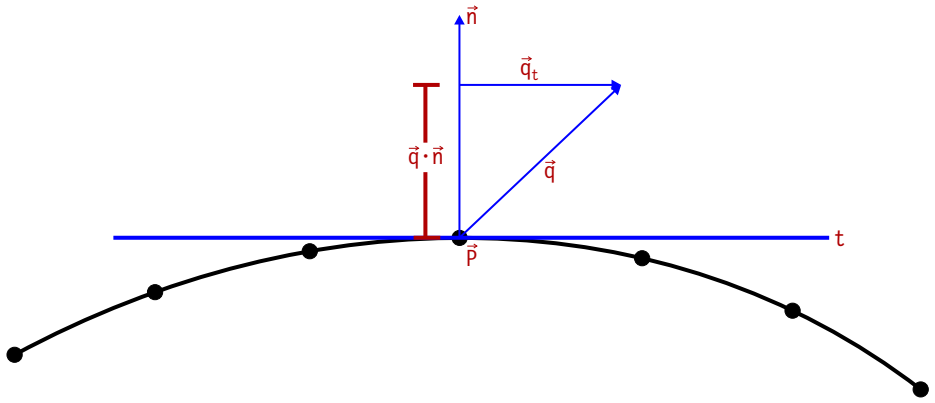


Abbildung 6: Geschwindigkeiten in einem Punkt des Oberflächennetzes

Der Normalenvektor ergibt sich aus der Normalisierung des Oberflächenvektors

$$\vec{n} = \frac{\vec{s}}{|\vec{s}|} \tag{7}$$

Laut [1] lässt sich die tangentiale Geschwindigkeit mit

$$\vec{q_t} = \vec{q} - (\vec{q} \cdot \vec{n}) \cdot \vec{n} \tag{8}$$

errechnen. Mittels der normalisierten tangentialen Geschwindigkeit kann der Betrag des Reibungsbeiwertes der freien Anströmung mit

$$\vec{c_f} = |\vec{c_{f_\infty}}| \cdot \frac{\vec{q_t}}{|\vec{q_t}|} \tag{9}$$

skaliert und somit der gesuchte Reibungsbeiwert im Punkt P berechnet werden. Der Reibungsbeiwert der freien Anströmung lässt sich laut [9] mit

$$c_{f_\infty} = \frac{\mu}{q_\infty} \tag{10}$$

ermitteln. Da auch dieser Wert innerhalb der vorliegenden Arbeit stets gegeben ist, wird dieser nicht näher erläutert.

3.3.3 Kräfte

Im Rahmen dieser Arbeit sollen für Profile jeweils die gesamte Druckkraft sowie die Reibungskraft bestimmt werden. Die Ermittlung der jeweiligen Kraft muss hierbei in jedem Punkt des Profils unabhängig vom umgebenen Netz erfolgen. Da die Anzahl der Punkte theoretisch unendlich ist, müssen die Kräfte in den tatsächlich vorhandenen Netzpunkten berechnet und aufsummiert werden.
Somit ergibt sich die gesamte Druckkraft aus der Summe der jeweiligen Produkte aus Druckbeiwert und Oberflächenvektor:

$$\vec{F_p} = -\sum_i c_{p_i} \cdot \vec{s_i} \tag{11}$$

Die Negierung dieser Summe ist wichtig, da der Oberflächenvektor $\vec{s}$ in einem Netzpunkt entgegen der angreifenden Druckkraft zeigt.

Die Reibung auf der Oberfläche eines Profils wirkt nicht nur in einem bestimmten Punkt P, sondern auch auf dessen umgebene Fläche, welche mit dem Betrag des Oberflächenvektors dargestellt wird. Daraus ergibt sich für die gesamte Reibungskraft:

$$\vec{F_f} = \sum_i \vec{c_{f_i}} \cdot |\vec{s_i}| \tag{12}$$

Die Gesamtkraft, welche auf ein Profil wirkt, ergibt sich aus der Summe der gesamten Druck- und der Reibungskraft:

$$\vec{F} = \vec{F_p} + \vec{F_f} \tag{13}$$

3.3.4 Auftrieb und Widerstand

Die Formeln für Auftrieb (L) und Widerstand (D) sind aus [9] entnommen. Sie lauten:

$$L = F_{up} \cdot \cos\alpha - F_{chord} \cdot \sin\alpha \tag{14}$$

$$D = F_{chord} \cdot \cos\alpha + F_{up} \cdot \sin\alpha \tag{15}$$

F_{up} entspricht dabei dem x-Wert des Gesamtkraftvektors, F_{chord} ist durch den y-Wert bestimmt. Der Winkel α ist in dieser Arbeit als gegeben zu betrachten.

3.3.5 Momente

Laut [1] berechnet sich ein Moment aus dem Vektorprodukt der angreifenden Kraft und einem Hebelarm in einem Punkt P.

$$\vec{M}_i = \vec{H}_i \times \vec{F}_i \tag{16}$$

Mittels eines gegebenen Bezugspunkts $\vec{B}$ lässt sich für jeden Netzpunkt der Hebelarm mit

$$\vec{H}_i = \vec{P}_i - \vec{B} \tag{17}$$

ermitteln. Durch das Aufsummieren der Momente erhält man das Gesamtmoment.

$$\vec{M} = \sum_i ((\vec{P}_i - \vec{B}) \times \vec{F}_i) \tag{18}$$

3.3.6 Schnittpunkte

Für die in dieser Arbeit geforderte Berechnung von Schnittlinien wird die Oberfläche eines Profils mit einer vorgegebenen Ebene geschnitten. Hierbei wird für jede Gerade zwischen zwei Netzpunkten der Schnittpunkt berechnet. Die Geradengleichung kann mit den Netzpunkten P und Q aufgestellt werden:

$$\vec{G} = \vec{P} + a \cdot \vec{r} \tag{19}$$

wobei $\vec{G}$ den Ortsvektor eines Punktes auf der Geraden darstellt und der Richtungsvektor $\vec{r}$ der Geraden mit

$$\vec{r} = \vec{Q} - \vec{P} \tag{20}$$

gegeben ist. Die Ebenengleichung in Koordinatenform mit dem Ortsvektor $\vec{R}$ lautet nach [1]:

$$d = \vec{R} \cdot \vec{n} \tag{21}$$

Der Ortsvektor $\vec{R}$ kann durch den Ortsvektor des Geradenpunktes und einem gegebenen Ortsvektor eines Punktes $\vec{E}$ der Ebene mit

$$\vec{R} = \vec{G} - \vec{E} \tag{22}$$

ersetzt werden. Da der Schnittpunkt der Geraden mit der Ebene gesucht wird muss der Abstand d dazwischen 0 betragen. Zusammengesetzt ergibt sich

$$0 = (\vec{P} + a \cdot (\vec{Q} - \vec{P}) - \vec{E}) \cdot \vec{n} \tag{23}$$

Somit kann der Skalierungsfaktor a mit

$$a = \frac{(\vec{E} - \vec{P})}{(\vec{Q} - \vec{P})} \tag{24}$$

ermittelt, dieser in Gleichung 19 eingesetzt und damit der Schnittpunkt berechnet werden.

3.3.7 Interpolation

Die Schnittpunkte liegen meist nicht in den Netzpunkten der Oberfläche sondern auf der Geraden zwischen zwei Punkten, siehe Abbildung 7.

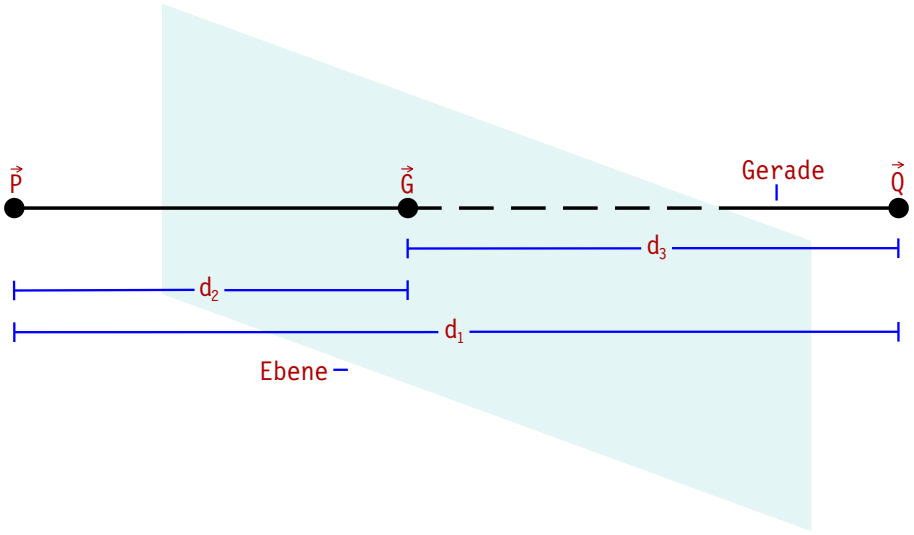


Abbildung 7: Interpolation von Daten

Für die Schnittpunkte, die nicht mit einem Netzpunkt identisch sind, müssen die vorhandenen Daten interpoliert werden. Dazu erfolgt, bezogen auf die Abbildung 7, die Berechnung der verschiedenen Abstände $d_i, i = 1, 2, 3$. Für den Abstand d_1 beispielsweise erfolgt die Aufstellung des Richtungsvektors $\vec{PQ}$ mit

$$\vec{PQ} = \vec{Q} - \vec{P} \tag{25}$$

Der Betrag dieses Richtungsvektors gibt den Abstand der beiden Punkte P und Q an:

$$d_1 = |\vec{PQ}| = \sqrt{(Q_x - P_x)^2 + (Q_y - P_y)^2 + (Q_z - P_z)^2} \tag{26}$$

Anschließend an die Berechnung der Abstände wird die relative Position des Schnittpunktes zu den Netzpunkten mit

$$PG = \frac{d_2}{d_1} \quad und \quad GQ = \frac{d_3}{d_1} \tag{27}$$

ermittelt. Mit diesen Werten lassen sich die Daten interpolieren. Für die Daten vom Schnittpunkt G (als G_D bezeichnet) gilt:

$$G_D = \frac{d_2}{d_1} \cdot Q_D + \frac{d_3}{d_1} \cdot P_G \tag{28}$$

4 Anforderungen

Dieses Kapitel beschreibt die Anforderungen, die sich aus der Aufgabenstellung ergeben. Sämtliche Anforderungen werden als Ziele an das zu entwickelnde Programm definiert. Des Weiteren folgt eine Schilderung der Arbeitsumgebung.

4.1 Zielsetzung

Eine einfache Bedienung bei gleichzeitig flexibler Einsetzbarkeit bilden die Grundvoraussetzungen an das Programm.

Um die Wartbarkeit und Erweiterungen an dem zu entwickelnden Auswerteprogramm zu ermöglichen, soll es aus einer Sammlung von Modulen bestehen. Diese Module verwalten die Daten und führen die darauf basierenden Berechnungen und Manipulationen möglichst allgemeingültig durch. Das heißt, es sollen sowohl strukturierte als auch unstrukturierte Daten verarbeitet werden können. Des Weiteren ist hierbei auf spätere Weiterentwicklungsmöglichkeiten des Auswerteprogramms zu achten. Die Grundlagen der Berechnungen wurden in Kapitel 3.3 besprochen.
Hinsichtlich der Erweiterung des Auswerteprogramms für Hubschrauberkonfigurationen ist die Skalierung einzelner Daten nötig. Damit lassen sich beispielsweise Koordinatentransformationen durchführen, die wichtig für Auswertungen an Rotorblättern von Hubschraubern sind.

Die Entwicklung des Programms soll in seiner Gesamtheit in der Programmiersprache Python erfolgen. Dies ermöglicht eine einfache Integration in bestehende Prozessabläufe des DLR. Dabei soll der Ablauf im Batch-Modus mittels einer Skript-Datei möglich sein. In solch einer Datei werden Befehle nacheinander aufgeführt, die wiederum eine sequentielle Aufgabenbearbeitung nach sich ziehen. Für die Auswertung einer Strömungssimulation soll der Benutzer dementsprechend mittels dieser Datei die erforderlichen Module auswählen und in gewünschter Reihenfolge ablaufen lassen können.

Aus der Aufgabenstellung geht hervor, dass das Auswerteprogramm modular aufgebaut und in Python entwickelt werden soll. Aufgrund dessen und der Möglichkeit zur Auswahl einzelner Module in einer Skript-Datei bietet sich zur Entwicklung die Objektorientierung an. Mit ihr lässt sich das Programm übersichtlich und zudem modular

aufbauen, wobei für den Benutzer eine so genannte API[1] zur Verfügung gestellt werden muss. Mittels dieser lassen sich einzelne Funktionalitäten aufrufen und damit ausführen. Zur Ausführungszeit nicht benötigte Funktionalitäten werden dementsprechend nicht aufgerufen.
Diese Vorgehensweise ist mit der prozeduralen Programmierung so einfach und übersichtlich nicht zu erreichen.

Die Auswertung von Daten erfolgt aufgrund der Ergebnisse beider Strömungslöser FLOWer und TAU. Dabei sind ausschließlich Oberflächendaten zu berücksichtigen. Die Auswertung von Feldgrößen (Daten im Raum um das Profil herum) findet in dieser Arbeit nicht statt.

4.2 Arbeitsumgebung

Die Entwicklung des Auswerteprogramms soll in der Programmiersprache Python erfolgen, die zu den höheren Programmiersprachen zählt. Das bedeutet, man kann mit ihr algorithmische Verfahren in einer rechnerunabhängigen problemorientierten Form beschreiben. [7]
Neben den höheren Programmiersprachen existieren beispielsweise so genannte Maschinensprachen. Diese sind speziell auf eine Maschine, einen Rechner, angepasst. Das heißt, in einer Maschinensprache entwickelte Programme können meist nicht ohne Veränderungen auf einem anderen Rechner ausgeführt werden.
Bei den höheren Programmiersprachen wie Python ist es nötig, den Quellcode vor der Ausführung in Maschinensprache zu übersetzen. Bei einigen Sprachen erfolgt diese Übersetzung erst zur Laufzeit des Programms, so dass sich die Ausführungszeit deutlich verlängert. Allerdings ist die Entwicklung von Programmen in höheren Sprachen wesentlich schneller, da deren Quellcode leichter zu lesen und zu verstehen ist.

Die Übersetzung von Quellcode in Maschinensprache kann einerseits mittels dem Kompilierer erfolgen. Hierbei wird das gesamte Programm kompiliert, in Maschinensprache übersetzt. Danach liegt es in ausführbarer Form vor. Andererseits existieren so genannte Interpreter, die den Quellcode meist Zeile für Zeile einlesen, übersetzen und sofort ausführen.
Interpretierte Programme sind im Gegensatz zu den kompilierten Programmen in ih-

[1] Application Programming Interface, Programmierschnittstelle

rer Ausführung meist langsamer. Der Vorteil der interpretierten Programme liegt darin, dass nur deren Interpreter an den Rechner angepasst werden muss. Im Gegensatz zu den kompilierten Programmen müssen an den interpretierten keine Änderungen erfolgen.

Für die Entwicklung in der Programmiersprache Python ist keine IDE[2] nötig. Statt dessen benötigt man nur einen Texteditor, wie es ihn in der Grundausstattung jedes Betriebssystems gibt. Des Weiteren ist vor allem das Programm zum Interpretieren des Quellcodes erforderlich. Im Rahmen dieser Arbeit wird für die Entwicklung die Python-Interpreter-Version 2.2.3 für das Betriebssystem SuSE Linux 9.0 eingesetzt, welche von der Internetseite [6] heruntergeladen werden kann. Damit lässt sich in der Konsolenumgebung des Betriebssystems eine Datei mit Python-Quellcode durch den Befehl *python pythonCodeDatei.py* ausführen.

Mit der Programmiersprache Python sind sowohl die prozedurale als auch die objektorientierte Programmierung möglich. [2] Im Hinblick auf die Aufgabenstellung wird in dieser Arbeit die Objektorientierung eingesetzt. Damit lässt sich die Entwicklung sehr gut modularisiert durchführen, außerdem wird dadurch die Wartbarkeit und Erweiterbarkeit des Programms erhöht.
Die Objektorientierung bietet die Möglichkeit Beziehungen zwischen Objekten abzubilden. Ein Objekt ist hierbei ein problemrelevanter Gegenstand der Betrachtung, welcher über verschiedene Eigenschaften verfügt, die so genannten Attribute. Das Verhalten des Objekts wird durch festgelegte Operationen (Methoden) beschrieben. Eine Klasse bildet dabei eine Art Schablone für ähnliche Objekte. Sie definiert für eine Sammlung solch gleichartiger Objekte deren Struktur (Attribute) sowie deren Verhalten mittels Methoden. [7] Die aus diesen Klassen erzeugten Objekte werden auch als Instanzen bezeichnet. Verschiedene Klassen können untereinander in unterschiedlicher Art und Weise in Beziehung stehen und somit auch deren Instanzen. Diese Beziehungen können durch so genannte Nachrichten zwischen den Instanzen realisiert werden. Eine Instanz ruft dabei eine Operation eines anderen Objekts auf. Es besteht somit eine Kommunikation zwischen den einzelnen Instanzen.
Der Einsatz der Objektorientierung im Rahmen dieser Arbeit wird im folgenden Kapitel 5 näher beschrieben.

[2] Integrated Development Environment, Entwicklungsumgebung

5 Programmentwurf

In diesem Kapitel wird die Arbeitsweise des zu entwickelnden Programms beschrieben. Des Weiteren werden für diese Diplomarbeit verwendete Hilfswerkzeuge vorgestellt und deren Einsatz erläutert. Im Anschluss daran folgt eine Vorstellung der Modellierung des Auswerteprogramms.

5.1 Programmbeschreibung

Das im Rahmen dieser Arbeit zu entwickelnde Auswerteprogramm soll laut Anforderungen in der Lage sein, sowohl strukturierte als auch unstrukturierte Ausgabedateien der Strömungssimulationsverfahren FLOWer und TAU einzulesen. Das gleichzeitige Umwandeln der Daten in ein einheitliches Format hat eine Vereinfachung der späteren Weiterentwicklung und der Programmbenutzung zur Folge.
Auf der Basis dieser Datenstruktur finden die in Kapitel 3.3 beschriebenen Berechnungen statt. Die daraus resultierenden Ergebnisse sind zum Einen die Kräfte und Momente, welche auf ein gesamtes Profil wirken. Zum Anderen besteht die Möglichkeit das Profil mit einer Ebene zu schneiden, wodurch am Schnitt des Profils mit der Ebene eine Linie entsteht. Für solch eine Schnittlinie sollen die Kräfte und Momente ebenso berechnet werden.
Anschließend an die Berechnungen ist der Benutzer in der Lage sich die Daten mit den berechneten Werten in Dateien ausgeben zu lassen. Abbildung 8 verdeutlicht diese Vorgehensweise.
Die Ausgaben des Auswerteprogramms müssen in einem der in Kapitel 3.2 beschriebenen Formate geschehen, da sie sonst nicht weiterverwendet werden können. Dies betrifft einerseits die Ausgabe der Profildaten, andererseits der Schnittliniendaten, jeweils mit den berechneten Werten für Kräfte und Momente. Da die Umwandlung von unstrukturierten in strukturierte Daten allgemein nicht möglich ist, außer in Sonderfällen, wird die gegensätzliche Methode verwendet. Dementsprechend liefert das Auswerteprogramm rein unstrukturierte Dateien.
Mit der einheitlichen Ausgabe ist ebenso begründet, dass für die Datenspeicherung während des Programmablaufs einheitlich eine unstrukturierte Struktur genutzt wird. Die Datenstruktur der strukturierten Dateien wird während des Einlesens in eine unstrukturierte umgewandelt, unstrukturierte Dateien können ohne großen Aufwand ein-

gelesen und zwischengespeichert werden.

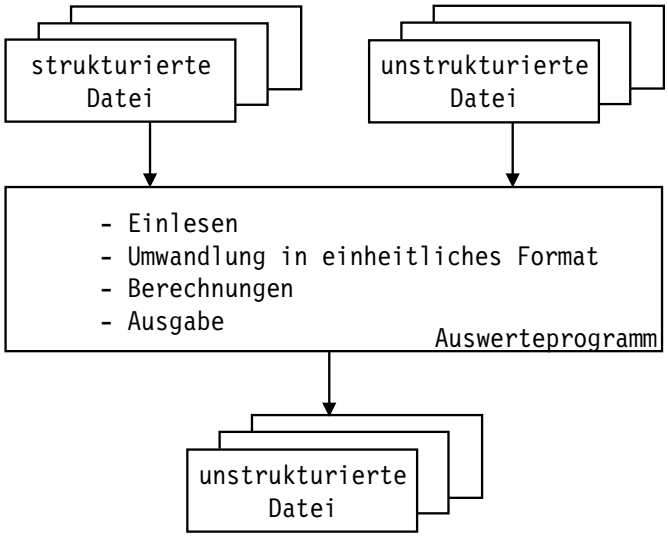


Abbildung 8: Programmablauf

Laut [11] wird bei einer Analyse ein zu untersuchendes Objekt in seine Bestandteile zerlegt und anschließend ausgewertet. Das in dieser Arbeit zu entwickelnde Programm verfährt nach diesem Prinzip. Es soll die Daten einer eingelesenen Datei anhand des Aufbaus aufgliedern und darauf verschiedene Berechnungen durchführen. Deshalb und aus dem Grund, dass es in der Programmiersprache Python entwickelt wird, setzt sich der Name des Auswerteprogramms aus den Bestandteilen „Analyse“ und „Python“ zusammen. Es wird im weiteren Verlauf dieser Arbeit auch als „Analython“ bezeichnet.

5.2 Hilfswerkzeuge

Im Rahmen dieser Diplomarbeit werden verschiedene Hilfswerkzeuge (Programme) eingesetzt. Sie werden in diesem Kapitel vorgestellt und deren Einsatz wird beschrieben.

5.2.1 Numpy

Der Name „Numpy“ setzt sich aus den Teilen Numerical und Python zusammen. Es bezeichnet eine Zusatzsoftware für Python. Der Name lässt darauf schließen, dass sie für die Handhabung mit Zahlen eingesetzt wird. Tatsächlich bietet die Software die Möglichkeit mit großen Datenfeldern, so genannten Arrays, schnell und effizient zu arbeiten.

Das Auswerteprogramm Analython soll in der Lage sein, sehr viele Datenzeilen, die wiederum aus mehreren einzelnen Werten bestehen, einzulesen und auf dieser Basis Berechnungen durchzuführen. Im Hinblick auf die hier beschriebene Entwicklung und spätere Weiterentwicklungen ist es notwendig, diese Daten einerseits übersichtlich zu speichern und zu verwalten. Andererseits steigern schnelle Berechnungen die Anwenderfreundlichkeit. Aus diesen Gründen wird in dieser Arbeit die Zusatzsoftware Numpy in der Version 23.1 angewendet, die im Internet kostenfrei unter [5] zu beziehen ist.

5.2.2 Tecplot

Im weiteren Verlauf dieser Arbeit werden mit dem zu entwickelnden Programm verschiedene Tests durchgeführt. Dabei müssen die Berechnungen von Schnittlinien beispielsweise visuell überprüft werden, da sich so die Position der Linie und deren Verlauf einfach kontrollieren lässt. Des Weiteren werden die Resultate der Entwicklung von Analython in dieser Arbeit beschrieben.
Für die visuellen Tests und die bildliche Ergänzung der Beschreibung wird das Programm Tecplot eingesetzt. Laut dem dazugehörigen Handbuch [8] ist es ein leistungsstarkes Werkzeug um eine große Auswahl an technischen Daten visualisieren zu können. Es bietet die verschiedensten Möglichkeiten zur Darstellung an. Dies sind beispielsweise zwei- oder dreidimensionale Oberflächenansichten oder eine dreidimensional, volumetrische Veranschaulichung.
Aufgrund dessen, dass es im DLR einen sehr breiten Einsatz erfährt und sich dabei sehr gut bewährt hat, wird es auch in dieser Arbeit verwendet. Des Weiteren bildet dieses Programm die Grundlage der Aufbauregeln für die hier verwendeten Ein- sowie Ausgabedateien, damit diese mit Tecplot visualisiert werden können.

5.2.3 Jumli

Jumli ist ein Programm, das zur Modellierung in der objektorientierten Programmierung und somit auch im weiteren Verlauf dieser Arbeit genutzt wird. Es bietet mittels der UML[3] die Möglichkeit unterschiedliche Diagrammtypen darzustellen, welche ein zu realisierendes Programm abbilden. Die von der OMG[4] entwickelte UML stellt

[3] Unified Modelling Language, Modellierungssprache
[4] Object Management Group

eine standardisierte Beschreibungssprache dar. Damit lassen sich die Strukturen und Abläufe für alle objektorientierten Softwaresysteme einheitlich beschreibend darstellen. Das Klassendiagramm dient beispielsweise zur Visualisierung der Struktur eines Systems. Es zeigt die Beziehungen der Klassen (siehe dazu Kapitel 4) untereinander auf. Das Sequenzdiagramm andererseits veranschaulicht den zeitlichen Aspekt. Es zeigt die chronologische Abfolge der Kommunikation während der Programmausführung.
Des Weiteren ist Jumli in der Lage aufgrund der Diagramme Quellcode zu generieren. Der objektorientierte Quellcode kann in der derzeitig verfügbaren Version nur für die Programmiersprachen C++ und Java erzeugt werden. Aus diesem Grund wird Jumli in der aktuellen Version 1.4 einzig für das Erstellen der Diagramme während der Programmgestaltung genutzt. Es kann kostenlos aus dem Internet unter [3] bezogen werden.

5.3 Programmgestaltung

Aufgrund verschiedener Aspekte, wie zum Beispiel der späteren Weiterentwicklung und da diese Entwickler und die Benutzer des Auswerteprogramms im DLR teilweise verschiedene Sprachen sprechen, wird die Implementierung (Kommentare, Beschreibungen) in der Standardsprache Englisch vollzogen. Hiermit wird eine allgemein verständliche Basis geschaffen.

Mittels des in Kapitel 5.2.3 beschriebenen Programms Jumli wird das Konzept des zu entwickelnden Auswerteprogramms modelliert. Mit den Diagrammtypen der UML kann der Programmaufbau und -ablauf schon vor der Implementierungsphase nachvollzogen und gegebenfalls angepasst werden. In diesem Kapitel wird der Aufbau des Programms anhand eines Klassendiagramms erläutert. Bei der Erstellung dieses Diagramms wird eine Einteilung (Klassifizierung) der Anforderungen durchgeführt. Das Verhalten der Instanzen wird durch verschiedene Methoden repräsentiert, die im anschließenden Kapitel 5.3.1 näher beschrieben werden.

Die in diesem Kapitel angelegten Klassen und deren Beziehungen untereinander werden in Form des Klassendiagramms in Abbildung 9 zur Verdeutlichung aufgezeigt.

Das Auswerteprogramm Analython soll eine übersichtliche Benutzerschnittstelle (API) besitzen, die den Zugriff auf alle Funktionalitäten des Programms bietet. Die API kontrolliert die Benutzereingaben auf potentielle Fehler und steuert die weitere Verarbei-

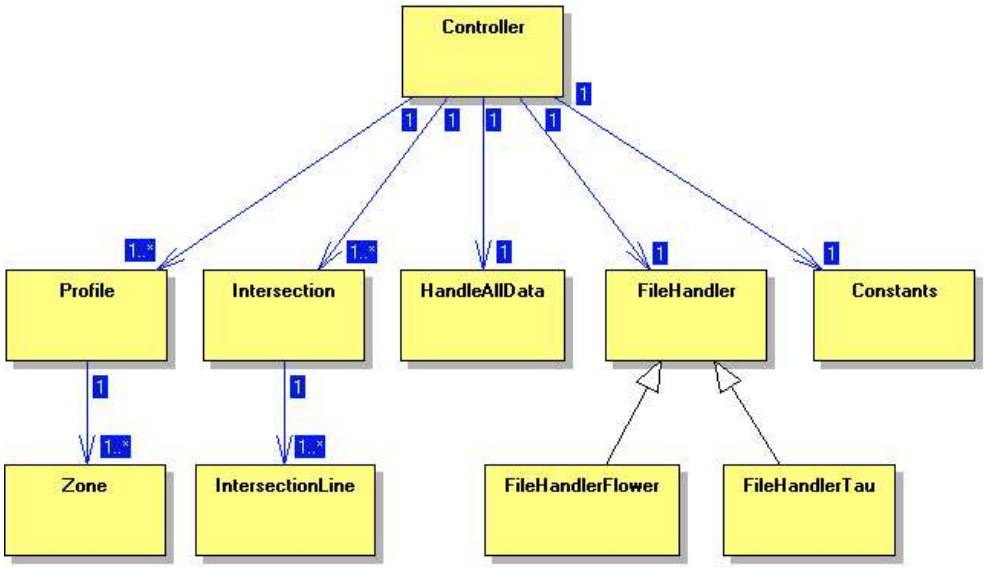


Abbildung 9: Aufbau des Klassendiagramms ohne Methoden

tung. Aufgrund dieser Steuerung und Überwachung wird diese Schnittstellenklasse im Weiteren als *Controller* bezeichnet.

Laut den Erläuterungen in den vorangegangenen Kapiteln soll Analython die Daten eines oder mehrerer Profile einlesen und darauf Berechnungen durchführen. Demzufolge existiert eine Klasse namens *Profile*. Die Instanz des Controllers gibt während der Programmausführung verschiedene Befehle an die bestehenden Instanzen von Profile. Demzufolge besitzt der Controller eine Beziehung zu mehreren Profile-Instanzen.

Ein Profil wiederum kann aus den Daten mehrerer Zonen bestehen. Folglich besitzt jeweils eine Profile-Instanz eine Beziehung zu mehreren *Zone*-Instanzen.

Des Weiteren soll Analython Schnittlinien berechnen können. Diese entstehen durch das Schneiden eines Profils mit einer vorgegebenen Ebene. Diese Linien können anschließend losgelöst vom Profil behandelt werden. Der Controller verwaltet demnach mehrere Schnittmengen (*Intersection*). Eine Schnittmenge besteht dabei aus mehreren Schnittlinien (*IntersectionLine*), jeweils eine Linie für den Schnitt mit einer Zone aus dem jeweiligen Profil.

Weiterhin besitzt der Controller eine einfache Beziehung zu einer Art Speicher an Variablen, deren Wert sich nicht ändert. Diese so genannten Konstanten werden in der Klasse *Constants* festgelegt, da dies übersichtlicher ist. Durch die zentrale Speicherung von Konstanten werden Änderungen sowie Weiterentwicklungen vereinfacht.

Außerdem muss Analython die Möglichkeit besitzen, Dateien auszulesen beziehungsweise schreiben zu können. Hierfür wird die Klasse *FileHandler* angelegt, die einheitliche Operationen, wie zum Beispiel das Öffnen von Dateien, beinhaltet. Von dieser Klasse erben zwei weitere Klassen *FileHandlerFlower* und *FileHandlerTau*. Durch die Vererbung besitzen diese beiden Klassen die Operationen von FileHandler und können dazu eigene Operationen aufweisen. Dies ist erforderlich, da FileHandlerFlower strukturierte Dateien, FileHandlerTau hingegen unstrukturierte Dateien auslesen kann und diese Klassen hierfür unterschiedliche Implementierungen benötigen.

Auf der Basis der eingelesenen Daten erfolgen verschiedene Berechnungen, wie zum Beispiel die Umwandlung von strukturierten in unstrukturierte Daten. Hierfür definiert die Klasse *HandleAllData* die notwendigen Operationen. Diese Klasse besitzt ebenfalls eine einfache Beziehung zum Controller.

5.3.1 Methodenbeschreibung

In diesem Kapitel werden für die Programmausführung wichtige Methoden aufgezeigt und deren Arbeitsweise kurz erläutert. Der gesamte Ablauf wird vom Controller ausgehend bis hin zu den einzelnen Klassen FileHandlerFlower, FileHandlerTau, Zone, IntersectionLine und HandleAllData erklärt. Die zu behandelnden Methoden sind im Klassendiagramm hinzugefügt und in der Abbildung 10 dargestellt.

Zu Beginn der Programmausführung müssen Daten aus einer Datei gelesen werden. Der Controller-Methode *readFileSaveData* muss übergeben werden, um welche Art von Datei, strukturiert oder unstrukturiert, es sich dabei handelt. Damit kann der Controller die jeweilige FileHandler-Klasse (Flower oder Tau) instanziieren und anschließend deren Methode *readFile* aufrufen. Über den Controller werden die ausgelesenen Daten an die Methode *saveNewData* der HandleAllData-Instanz weitergeleitet. Diese speichert alle Daten zwischen und wandelt gleichzeitig die strukturierten Daten in unstrukturierte Daten um. Letztendlich berechnet die HandleAllData-Methode *calcSurfaceVectors* alle Oberflächenvektoren.

Mit *createNewProfile* erstellt der Controller eine neue Instanz der Klasse Profile mit der zu übergebenden Zeichenkette als Namen, die für das Profil eindeutig sein muss. Die Methode *addZonesToProfile* bewirkt, dass alle oder nur einzelne Zonen und somit deren Daten aus dem Zwischenspeicher zum neuen Profil hinzugefügt werden. Dazu

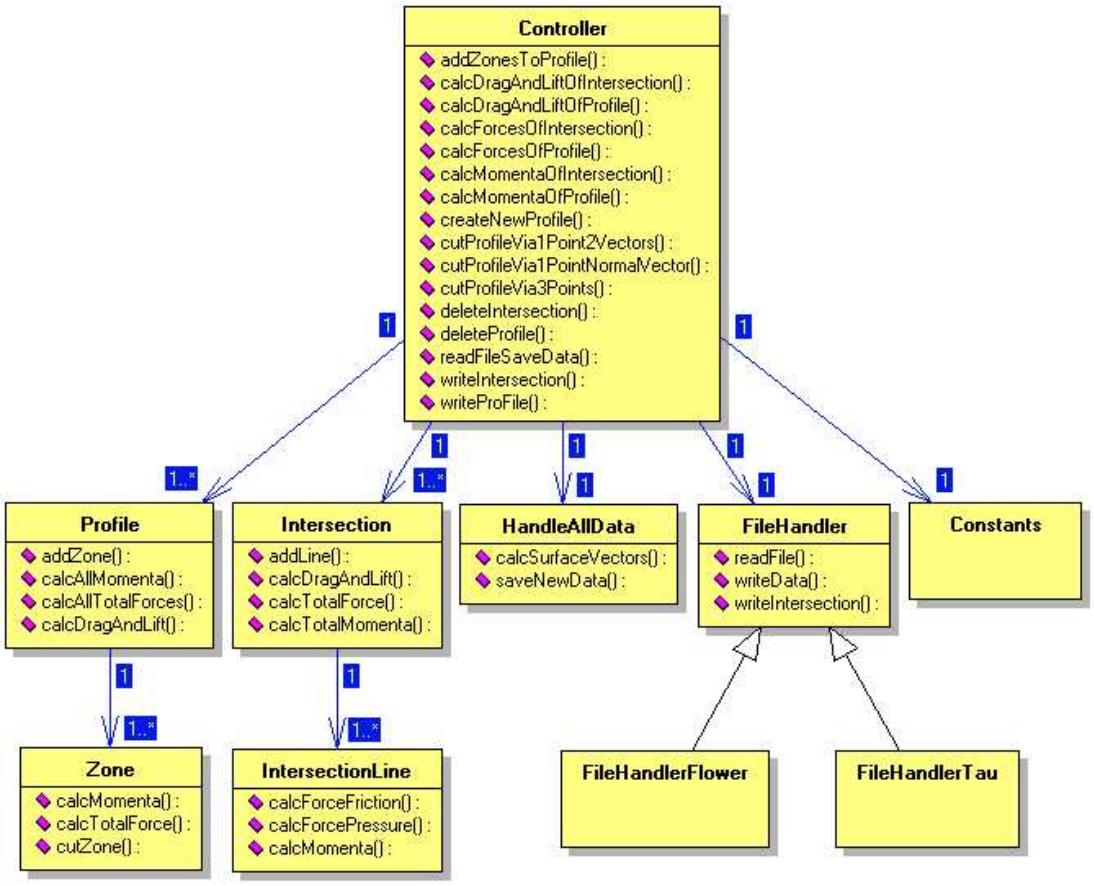


Abbildung 10: Aufbau des Klassendiagramms mit den wichtigsten Methoden

wird für jede Zone die Profile-Methode *addZone* aufgerufen, die wiederum jeweils eine Instanz der Klasse Zone erzeugt und die Daten gleichzeitig zur Speicherung an die neue Zone-Instanz übergibt.

Die drei Methoden *calcForcesOfProfile*, *calcMomentaOfProfile* und *calcDragAndLiftOfProfile* des Controllers geben den Aufruf an die ähnlich lautenden Methoden *calcAllTotalForces*, *calcAllMomenta* und *calcDragAndLift* der benannten Profil-Instanz weiter. Die ersten beiden Methoden von Profile geben wiederum diesen Aufruf an alle Zone-Instanzen weiter (*calcTotalForce, calcMomenta*). Dadurch berechnen diese Instanzen für deren Daten die gesamten Kräfte und Momente. Die Profile-Methode *calcDragAndLift* hat zur Folge, dass der Auftrieb und der Widerstand für das Profil berechnet werden.

Mittels der Controller-Methode *writeProFile* lässt sich ein vorher angelegtes Profil in einer spezifizierten Datei speichern. Über so genannte get-Methoden, die in Abbildung 10 der Übersichtlichkeit wegen nicht aufgeführt sind, bekommt der Controller die Daten des Profils. Die Profile-Instanz wiederum gelangt ebenso über get-Methoden an die Daten

der einzelnen Zonen. Zusätzlich werden auch die vorher berechneten Daten wie Kräfte und Momente in dieser Weise übermittelt. Die gesamten Daten werden vom Controller an die Methode *writeData* einer neu erzeugten FileHandler-Instanz übergeben, die diese in die gewünschte Datei schreibt.

Die Mehoden *cutProfileVia1Point2Vectors*, *cutProfileVia1PointNormalVector* und *cutProfileVia3Points* dienen jeweils zur Berechnung der Schnittmenge eines Profils mit einer angegebenen Ebene. Die Ebene kann mittels der drei Methoden auf unterschiedliche Weise definiert werden, je nachdem wie der Benutzer es wünscht. Je nach Methode kann diese Ebene in Parameterform, Hessescher Normalform oder mit 3 Punkten, die in der Ebene liegen, festgelegt werden. Diese Daten werden der Methode *cutProfile* des jeweiligen Profils übergeben, die wiederum *cutZone* aller Zone-Instanzen damit aufruft. Der Controller legt eine neue Instanz der Intersection-Klasse an. Mittels einer get-Methode gelangt der Controller an die Linien-Daten des Profils, das seinerseits die Daten der Zone bekommt. Die Linien-Daten jeweils einer Zone werden der Methode *addLine* der neuen Intersection-Instanz übergeben. Diese Instanz legt für jede Linie eine neue IntersectionLine-Instanz an und übergibt gleichzeitig alle Daten.

Ähnlich wie bei den Profilen lassen sich auch mit den Schnittmengen, bestehend aus den Schnittlinien, Kräfte, Momente und so weiter berechnen. Mittels der Controller-Methode *writeIntersection* lässt sich eine gewünschte Schnittmenge in einer Datei speichern.

Für die Erweiterung des Auswerteprogramms an Hubschrauberkonfiguration ist es beispielsweise notwendig einige weitere Methoden zu implementieren, mit denen man die Daten der Profile und der Schnittlinien skalieren kann. Die Skalierung von Daten wird zum Beispiel notwendig, wenn der Benutzer Koordinatentransformationen durchführen möchte. Die Methoden hierzu sind in Abbildung 10 nicht aufgeführt, da das Diagramm sonst zu unübersichtlich wird.

Abschließend soll sich dem Benutzer die Möglichkeit bieten, vorher angelegte Profile und Schnittmengen zu löschen, um beispielsweise deren Bezeichnung wiederzuverwenden und damit nicht weiter benötigte Daten aus dem Speicherbereich gelöscht werden. Die Methoden *deleteProfile* und *deleteIntersection* übernehmen diese Aufgabe.

6 Implementierung

Während der Implementierungsphase werden die einzelnen Module von den Grundanforderungen, wie Dateneingabe, Speicherung und deren Ausgabe, bis hin zu den speziellen Anforderungen, den Berechnungen auf der Grundlage der Daten, entwickelt. Diese Vorgehensweise wird in diesem Kapitel in Bezug auf die einzelnen Module näher erläutert. Dabei wird auf verschiedene Problemstellungen während der Entwicklung eingegangen und deren Lösungen aufgezeigt.

6.1 Eingabe, Speicherung, Ausgabe

Die erste Aufgabe während der Implementierungsphase besteht darin, die Eingabe von Daten, deren Zwischenspeicherung und Ausgabe zu realisieren. Die hierfür beteiligten Klassen sind in Abbildung 11 nochmals dargstellt.

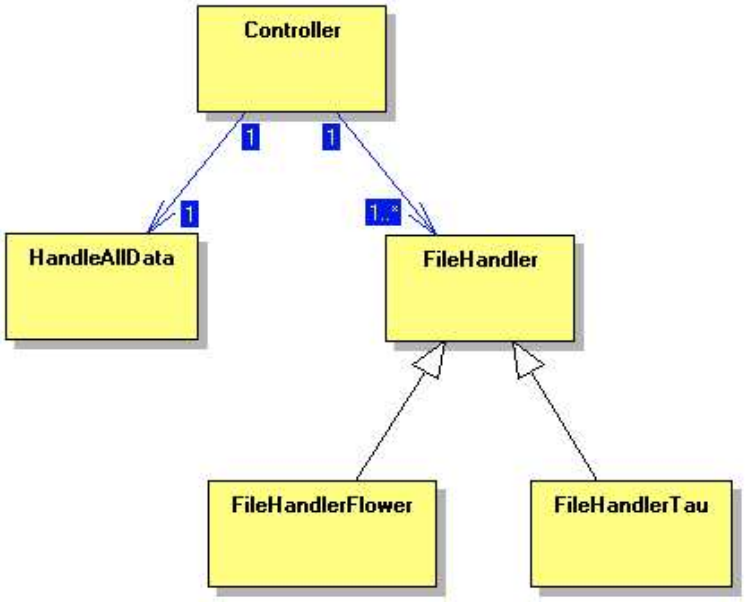


Abbildung 11: Klassendiagrammausschnitt für die Eingabe, Speicherung und Ausgabe

Für die Eingabe von Daten müssen Methoden der Klassen Controller, FileHandler, FileHandlerFlower und FileHandlerTau implementiert werden.
Die Klasse FileHandler wird als abstrakt deklariert, da sie gleiche Methoden für die Klassen FileHandlerFlower sowie FileHandlerTau beinhaltet und gleichnamige Methoden ohne Implementierung (Abstraktion) definiert. Diese abstrakten Methoden werden in FileHandlerFlower und FileHandlerTau jeweils unterschiedlich implementiert. Dies

betrifft einerseits die Analyse der Profildeklaration in der einzulesenden Datei. Für strukturierte Daten muss der Geschwindigkeitsvektors $\vec{q}$ vorhanden sein, der durch die Variablen u, v und w repräsentiert wird, damit sich daraus der Reibungsbeiwert c_f brechnen lässt. In Dateien mit unstrukturierten Daten ist dieser Reibungsbeiwert stets gegeben. Andererseits betrifft es die Analyse der Zonendeklarationen wie sie in Kapitel 3.2 beschrieben sind. Für strukturierte Dateien müssen beispielsweise die Parameter I und J, bei unstrukturierten Dateien die Parameter N und E selektiert werden. Aus diesen Gründen erfolgt auch das Einlesen der Daten unterschiedlich. Während für strukturierte Daten nur die Werte der einzelnen Parameter Zeile für Zeile ermittelt werden, erfolgt bei unstrukturierten Daten daraufhin das Einlesen der Verbindungsdaten, die angeben, wie die Datenpunkte untereinander in Verbindung stehen. Die gemeinsamen Methoden beider Klassen hingegen beziehen sich auf das Öffnen und Schließen von Dateien.
Für das Einlesen der Daten und die darauf folgende Speicherung wird in der Controller-Klasse eine Methode implementiert, der übergeben wird, ob es sich um eine strukturierte oder unstrukturierte Datei handelt. Auf dieser Basis wird die entsprechende FileHandler-Klasse (Flower oder Tau) instanziiert und die Methode für das Auslesen aufgerufen.

Über die Controller-Instanz gelangen die selektierten Daten an die HandleAllData-Instanz. Diese Klasse definiert verschiedene Methoden für die Zwischenspeicherung aller eingelesenen Daten. Bei neuen Daten mit strukturiertem Format erfolgt mittels dieser Methoden die sofortige Umwandlung in ein unstrukturiertes Format. Hierbei werden der Reibungsbeiwert c_f berechnet, anschließend die Verbindungen der Netzpunkte untereinander ermittelt und gespeichert. Daraufhin liegen im Zwischenspeicher, der durch die Attribute der HandleAllData-Klasse repräsentiert wird, nur noch unstrukturierte Daten vor, für die zum Schluss alle Oberflächenvektoren mittels der in Kapitel 3.3.1 vorgestellten Gleichungen berechnet werden.
Bei dieser Speicherung wird zum ersten mal das Python-Zusatzpaket Numpy (siehe Kapitel 5.2.1) genutzt. Damit lassen sich die Daten übersichtlich in einem so genannten Array speichern, dass wie eine Matrix zeilen- und spaltenweise aufgebaut ist. Zusätzlich zu den in Kapitel 5.2.1 besprochenen Vorteilen von Numpy gehört weiterhin, dass ein solches Array schnell mittels des Befehls *resize()* in der Größe geändert werden kann. Es ist demnach nicht notwendig, am Anfang des Quellcodes die Größe des Arrays fest-

zulegen. Diese Möglichkeit ist wichtig, damit im weiteren Programmablauf zusätzliche Dateien eingelesen und somit Daten gespeichert werden können.
Des Weiteren werden ebenso die Zonenbeschreibungen abgespeichert, die in Kapitel 3.2 erläutert wurden. Ein Array ist für diese Speicherung ungeeignet, da es nur Zahlenwerte gleichen Typs (Ganzzahlen oder Fließkommazahlen) aufnehmen kann. Eine Zonenbeschreibung besteht allerdings aus Zahlen und Zeichenketten, zum Beipiel aus den Werte I und J für strukturierte Daten und dem Zonentitel. Für die Erfassung dieser Beschreibungen werden so genannte Listen genutzt. Diese erlauben das aufeinanderfolgende Speichern von Parameterwerten unterschiedlichen Typs. Im vorangegangenen Beispiel bedeutet dies, dass erst der Wert für I, danach für J und anschließend der Titel der ersten Zone hintereinander in dieser Liste festgehalten werden. Bei mehreren Zonen werden die Beschreibungen jeweils hinten an die Liste angehängt. Mit einer speziell angelegten Variable, die die Anzahl an gespeicherten Zonen repräsentiert, lässt sich somit auf jedes gesuchte Element zugreifen.

Die Ausgabe der Daten erfolgt wiederum mittels der FileHandler-Methoden. Die Klasse FileHandler definiert hierfür die benötigten Methoden. Da diese aber als abstrakte Klasse aufgrund unvollständiger Methoden definiert ist, ist es nicht möglich eine Instanz davon zu bilden. Statt dessen kann hierfür eine Instanz von FileHandlerFlower oder von FileHandlerTau angelegt werden, die die Methoden von FileHandler vererbt bekommen haben. Diesen Methoden, zum Ausschreiben der Daten in eine Datei, müssen alle benötigten Daten übergeben werden.

6.2 Berechnung von Kräften und Momenten

Die in diesem Kapitel betrachteten Berechnungen beziehen sich außer auf Kräfte und Momente weiterhin auf den Gesamtwiderstand und den Auftrieb, die mittels der vorher ermittelten Kräfte berechnet werden können. Die an diesen Berechnungen beteiligten Klassen sind in Abbildung 12 nochmals aufgezeigt.

Zum Einen lassen sich die Druckkraft, die Reibungskraft und die daraus resultierende Gesamtkraft für ein gesamtes Profil ermitteln. Über die Controller-Instanz wird diese Berechnung, deren Formeln in den Kapiteln 3.3.3 und 3.3.5 erläutert wurden, gestartet und der Befehl an die jeweilige, anzugebende Profile-Instanz weitergeleitet. Diese wiederum gibt den Befehl an alle Zone-Instanzen weiter, auf die sie eine Referenz besitzt.

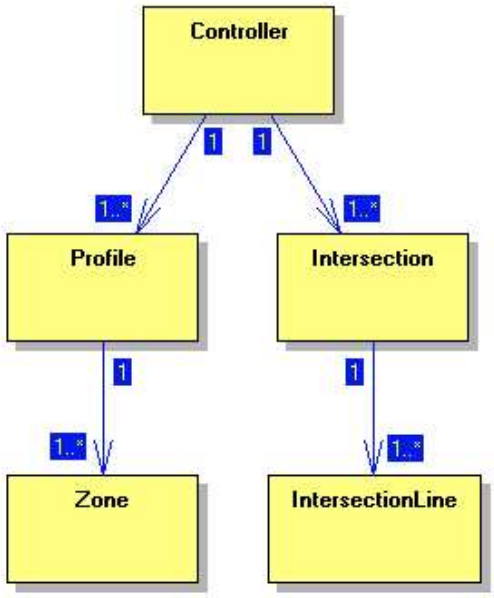


Abbildung 12: Klassendiagrammausschnitt für die verschiedenen Berechnungen

Dieselbe Vorgehensweise wird bei der Berechnung der Momente verfolgt.
Dadurch, dass jede Zone-Instanz die eigenen Kräfte und Momente berechnet und die Profile-Instanz diese addiert, ist der Quellcode übersichtlich gehalten und kann einfach verändert beziehungsweise erweitert werden.

Zum Anderen können Kräfte und Momente auch für die vorher ermittelten Schnittlinien berechnet werden. Hierfür wird ohne Unterschied vorgegangen. Der Befehl vom Controller wird über die jeweilige Intersection-Instanz an alle IntersectionLine-Instanzen weitergeleitet.

Auf der Basis der vorher berechneten Kräfte können mit Hilfe der in Kapitel 3.3.4 aufgeführten Gleichungen der Gesamtwiderstand des Profils oder der Schnittlinie sowie deren Auftrieb ermittelt werden. Am Beispiel der Profile addiert die Profile-Instanz alle Zonenkräfte und kann diese Gesamtkraft für die Berechnung von Widerstand und Auftrieb nutzen.

6.3 Schnittlinienberechnung

Bei der Berechnung von Schnittlinien kommen die gleichen Klassen zum Einsatz wie bei den Kräfteberechnungen, also die Klassen Controller, Profile, Zone, Intersection und IntersectionLine.

Für eine Schnittlinienermittlung legt die Controller-Instanz eine Instanz der Intersection-Klasse an und gibt den Befehl an das jeweilige Profil, dass dafür die Linie zu berechnen ist. Die Instanz der Profile-Klasse gibt diesen Befehl nacheinander an alle Zonen weiter. Jede Zone liefert daraufhin eine zusammenhängende Linie (Schnittpunkte mit der Ebene und deren Daten), die einzeln über den Controller an die Intersection-Instanz übergeben werden. Für jede Linie wird eine IntersectionLine-Instanz angelegt. Somit kann ein Schnitt über das jeweilige Intersection-Objekt und die darin enthaltenen, einzelnen Linien über IntersectionLine-Objekte angesprochen werden.

Zu Beginn dieser Arbeit sah das erste Konzept vor, dass die Zonen die Linien berechnen und die jeweilige Profile-Instanz die gesamten Daten als eine zusammenhängende Linie zusammenknüpft. Diese Daten wären dann ohne die Intersection-Klasse in einer IntersectionLine-Instanz gespeichert. Dieses Konzept brachte allerdings Probleme, insbesondere mit konkaven (nach innen gewölbten) beziehungsweise nicht zusammenhängenden Zonen mit sich. Im Gegensatz dazu sind konvexe Zonen nach außen gewölbt. Abbildung 13 zeigt eine konkave (A) und eine konvexe (B) Zone in zweidimensionaler Ansicht mit angedeuteten Netzpunkten. Beide Zonen besitzen keinen Verbindungspunkt zueinander.

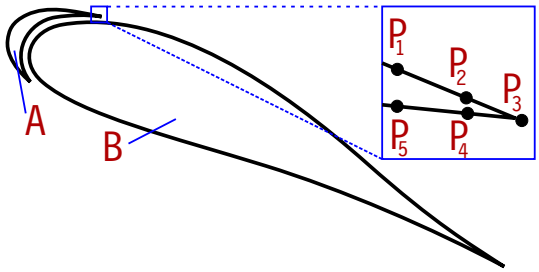


Abbildung 13: Konkave und konvexe Zone im Zweidimensionalen

Das erste Konzept sah vor, dass die Schnittlinie der Zone mit einer gegebenen Ebene wie folgt ermittelt werden soll:

1. Für alle ermittelbaren Geraden:
    - Bestimmen einer Geraden durch zwei benachbarte Netzpunkte
    - Berechnen des Schnittpunktes der Geraden mit der Ebene
    - Ermitteln, ob der Schnittpunkt zwischen den Netzpunkten liegt

2. Für alle Schnittpunkte:

    - Verbinden mit dem nächstgelegenen Schnittpunkt

Das Verbinden der Schnittpunkte in Stichpunkt 2 sollte darauf basieren, dass zwischen dem ausgehenden Punkt und allen weiteren der Abstand berechnet wird. Der nächstgelegene Punkt muss theoretisch derjenige mit dem kleinsten Abstand sein.
Allerdings hat sich herausgestellt, dass sehr häufig ein anderer Fall eintritt. Wie auf der rechten Seite in Abbildung 13 nochmals verdeutlicht, könnten bei sehr spitz zulaufenden Kanten die falschen Punkte verbunden werden. In der Beispielabbildung besitzt der Punkt P_4 zum Bezugspunkt P_2 einen kleineren Abstand als der richtige Punkt P_{P3}.

Des Weiteren kann, in Bezug auf die Beispielabbildung 13, mit der oben angesprochenen Vorgehensweise keine zusammenhängende Schnittlinie erzeugt werden, da beide Zonen keinen gemeinsamen Punkt haben. Eine theoretische Lösung hierfür wäre, dass nur eine Linie für eine Zone ermittelt und die andere wegfallen würde. Weiterhin ist es möglich die beiden Zonen an jeweils einem Punkt miteinander zu verbinden, beispielsweise vom Punkt P_3 der Zonenkante von A zum nächstgelegenen Punkt der Zone B.
Beide Varianten sind ungenügend, da sie die Schnittlinien nicht korrekt berechnen und somit die darauf basierenden Berechnungen von Kräften, etc. verfälschen.

Aufgrund der angesprochenen Schwierigkeiten wurde ein weiteres Konzept entwickelt. Dieses basiert darauf, dass ein Schnitt durch ein Profil insgesamt als Schnittmenge (Intersection) angesehen wird, die wiederum alle Schnittlinien (IntersectionLine) beinhaltet. Somit ist es möglich, für jede Zone einzeln die Schnittlinie mit der gegebenen Ebene zu berechnen und zu speichern.
Das Problem mit der Verbindung zwischen zwei nicht zusammenhängenden Zonen tritt somit nicht mehr auf. Stattdessen werden die verschiedenen Berechnungen für Kräfte und so weiter auf den einzelnen IntersectionLine-Instanzen durchgeführt, deren Ergebnisse die Intersection-Instanz zum Beispiel nur noch addieren muss.
Des Weiteren wird die Berechung der Schnittlinie für eine Zone ab Punkt 2 anders ausgeführt:

2. Für alle Polygone des Netzes (vergleiche hierzu Kapitel 3.1):

    - Ermittlung aller Schnittpunkte
    - Sortierung der Schnittpunkte nach Abständen

- Aufstellen der Teilschnittlinie

3. Bestimmen einer Start-Teilschnittlinie als Gesamtschnittlinie
4. Solange weitere Teilschnittlinien vorhanden:
    - Suchen von an der Gesamtschnittlinie angrenzenden Teilschnittlinien (davor und dahinter) anhand der Punktekoordinaten
    - Anfügen der Linien zur Gesamtschnittlinie

Hiermit werden einerseits die möglichen, konkaven Polygone mit berücksichtigt. Weiterhin werden die Schnittlinien zu einer gesamten Linie aufgrund der eindeutigen Koordinaten korrekt zusammengefügt.

6.4 Skalierung von Daten

Im Hinblick auf eine spätere Weiterentwicklung ist es notwendig, die eingelesenen und die berechneten Daten skalieren zu können. Speziell für die Auswertung von Kräften an sich drehenden Rotorblättern eines Hubschraubers ist die Skalierung nötig, da somit beispielsweise die Koordinaten transformiert werden können, damit das Koordinatensystem für ein Rotorblatt ständig denselben Bezug darauf besitzt.
Wie auch bei der Kräfteberechnung sind bei der Skalierung von Daten die Klassen Controller, Profile, Zone, Intersection und IntersectionLine beteiligt. Um die Übersichtlichkeit zu bewahren, existiert für jede Art von Größe (Zusammenfassung von Parametern nach deren Bedeutung - Koordinaten, Kräfte, und so weiter) eine Skalierungsmethode. Somit wird nicht für jeden Parameter eine Methode genutzt, denn dadurch würde die Anzahl an Skalierungsmethoden zu groß sein. Außerdem besteht die Möglichkeit für alle Werte eine einzige Skalierungsmethode zu nutzen. Dies wäre allerdings für den Benutzer unübersichtlich, da es mehr als zehn Parameter zu skalieren gibt. Die eingesetzte Skalierung ist unterteilt in die Skalierung der Netzpunktkoordinaten, Oberflächenvektoren, Beiwerte, Kräfte, Momente sowie von Auftrieb und Widerstand.

7 Test des Programms

Dieses Kapitel beschreibt die Durchführung der Programmtests sowie deren Resultate. Hierfür wurden verschiedene Testdateien zur Verfügung gestellt, mit denen die unterschiedlichen Funktionalitäten von Analython überprüft werden können. Die Kontrolle der Auswertung erfolgt mit Hilfe weiterer Dateien, die aufgrund früherer Berechnungen mit anderen Programmen die zu erhaltenden Ergebnisse beinhalten.

7.1 L1T2 Hochauftriebskonfiguration

Die Konfiguration L1T2 beinhaltet die Daten für einen Teil eines Flügels, der aus mehreren, strukturiert formatierten Zonen besteht. Dies wird in Abbildung 14 durch die farblich unterschiedliche Kennzeichnung deutlich.

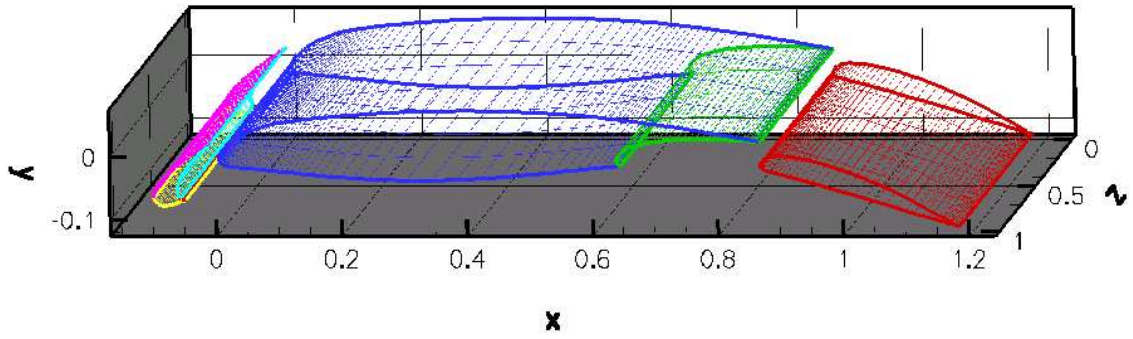


Abbildung 14: vorliegende Daten L1T2

An diesem Testbeispiel soll gezeigt werden, dass Analython die Kräfte und Momente eines gesamten Proils, als auch einer Schnittlinie korrekt berechnet. Die Lage der Schnittebene parallel zur Ebene, die durch die x-Achse und y-Achse des Koordinatensystems aufgespannt wird, spielt hierfür keine Rolle, da der Flügelausschnitt die Breite 1 besitzt, ebenso wie eine Schnittlinie (vergleiche hierzu Kapitel 3.3.1). Aufgrund dessen müssen die Kräfte und Momente des Profils und der Schnittebene identisch sein.

7.2 M6 Konfiguration

Für den Nachweis der richtigen Berechnung von Kräften und Momenten sowie der korrekten Ermittlung von Schnittlinien steht die M6 Konfiguration zur Verfügung. In Abbildung 15 wird der M6 Flügel mit strukturiertem Format dargestellt.

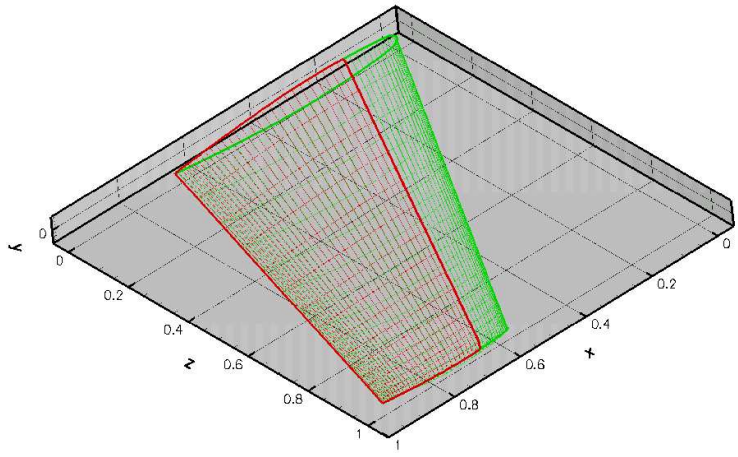


Abbildung 15: vorliegende Daten M6

Für den Nachweis, dass die Daten der Schnittlinien korrekt berechnet werden, sind die Beiwerte c_f sowie c_p als Pfeile in den Netzpunkten dargestellt, die den jeweiligen Richtungsvektor repräsentieren.

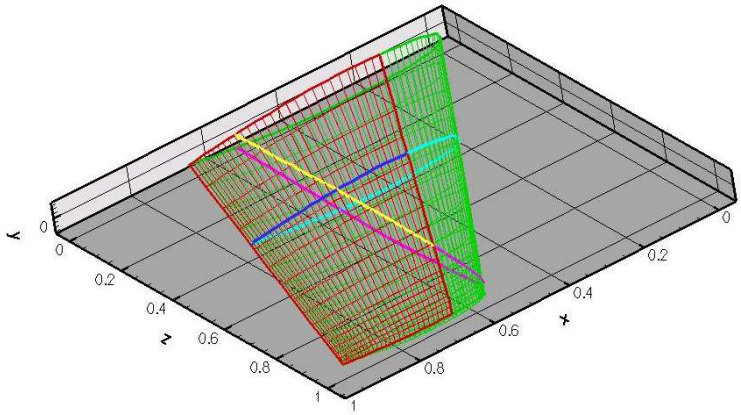


Abbildung 16: Ergebnis des M6-Tests (Schnittlinien)

xxx

7.3 Hirett Konfiguration

Die Hirett-Konfiguration stellt die Ergebnisdaten der Strömungssimulation für einen Teil eines Flugzeugrumpfes sowie einen daran befestigten, kompletten Flügel dar. Hierbei liegen die Daten im unstrukturierten Format vor. Die Netzpunkte sind so untereinander vernetzt, dass sich daraus Dreiecke bilden. Das Oberflächennetz hierfür ist in Abbildung 18 dargestellt.

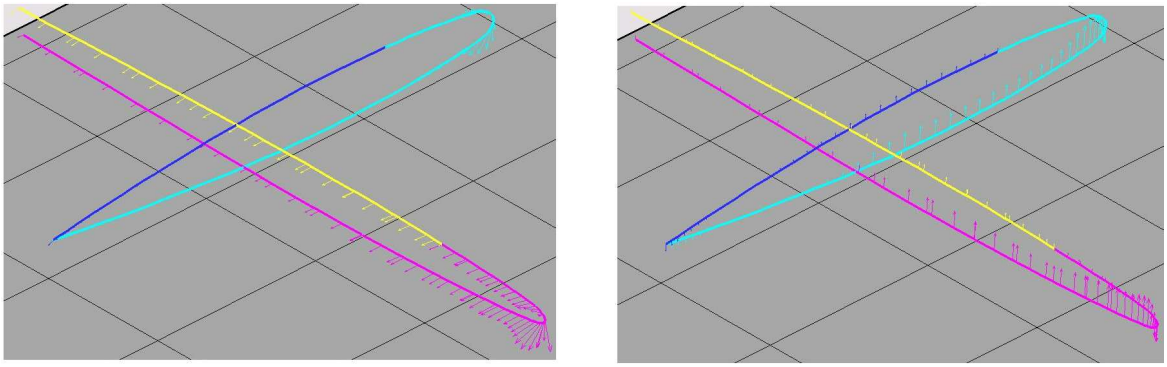


Abbildung 17: Ergebnis des M6-Tests (c_f und c_p)

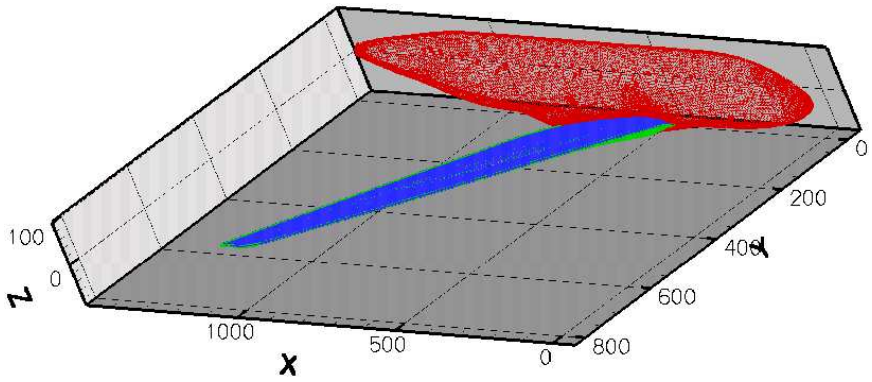


Abbildung 18: vorliegende Daten Hirett

Im ersten Schritt soll während dieses Tests gezeigt werden, das Kräfte und Momente von Analython richtig berechnet werden. Hierzu werden alle drei in der Abbildung 18 ersichtlichen Zonen (farblich unterschiedlich gekennzeichnet) von Analython eingelesen und für die gesamte Konfiguration die Kräfte sowie die Momente berechnet. Zur genaueren Kontrolle erfolgt hierbei die Aufspaltung der Kräfte in Druck-, Reibungs- und die Gesamtkraft.

Weiterhin soll gezeigt werden, dass Analython Schnitte durch ein Profil fehlerfrei ermittelt und die Daten an den Schnittpunkten richtig berechnet. Dafür werden zwei beliebige Schnittebenen ausgewählt, welche zum Einen den Flügel längs und zum Anderen den Flügel quer schneiden. Damit kann gezeigt werden, dass die Lage der Schnittebene in Bezug auf das Profil keine Rolle spielt und die Schnittlinien immer exakt erzeugt werden

Im letzten Schritt für diese Konfiguration wird ermittelt, ob mittels Analython viele Schnittlinien und deren Kräfte nacheinander berechnet und die gesamten Auftriebs- und Widerstandswerte, für eine spätere Visualisierung mittels Tecplot, in einer Da-

tei ausgegeben werden können. Dazu ist die Skriptdatei zum Start von Analython insofern angepasst, dass die Schnittlinienberechnung 101 Mal aurgerufen wird. Dabei ändert sich lediglich der y-Wert der Schnittebene, so dass entlang des Flügels mehrere, parallel zueinander liegende Schnittebenen definiert werden, die den gleichen Abstand voneinander besitzen. Für alle Schnittlinien werden die Kräfte und darauf basierend der Auftrieb und der Widerstand berechnet. Anschließend erfolgt die Ausgabe aller Auftriebs- und Widerstandswerte in einer gemeinsamen Datei mittels der Controller-Methode *writeDragAndLiftOfAllIntersections*. Diese Datei kann anschließend mittels Tecplot ohne weitere Veränderungen visualisiert werden.

7.4 DAUROT Hubschrauberkonfiguration

Eine Datei der DAUROT Hubschrauberkonfiguration beinhaltet Daten zu insgesamt sieben Zonen, drei Zonen für den Rumpf eines Hubschraubers und jeweils eine Zone für ein Rotorblatt, wie sie in Abbildung 19 abgebildet sind.

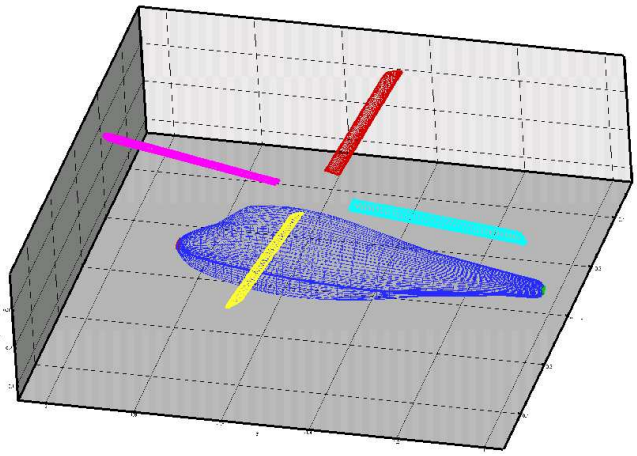


Abbildung 19: vorliegende Daten DAUROT

Die Daten der Hubschrauberkonfiguration sind in einem strukturierten Format abgespeichert. Im Gegensatz zu allen anderen Testfällen stellen diese Daten die Ergebnisse einer reibungslosen Strömungssimulation dar. Daraus folgt, dass der Reibungsbeiwert c_f (vergleiche hierzu Kapitel **??**) für alle Netzpunkte einen Wert von 0 besitzt.
Des Weiteren besteht die DAUROT Hubschrauberkonfiguration aus mehreren gleichartigen Ergebnisdateien, lediglich die Werte der einzelnen Parameter unterscheiden sich voneinander. Der Grund hierfür ist, dass jede Datei die Ergebnisse der Strömungssimulation für einen anderen Stand der Rotorblätter in Bezug auf den Hubschrauberrumpf

beinhaltet. Die DAUROT Hubschrauberkonfiguration stellt demnach die Ergebnisse einer kompletten Drehung (360°) der Rotorblätter in mehreren Ausgabedateien dar.

In diesem Testfall werden die Kräfte und Momente berechnet, die am Rumpf des Hubschraubers während einer Umdrehung der Rotorblätter wirken. Hierfür ist es allerdings nur notwendig, die Daten für eine viertel Umdrehung auszuwerten, da sie sich anschließend für jedes weitere Rotorblatt periodisch wiederholen. Die Strömungssimulationsergebnisse für eine viertel Umdrehung sind insgesamt in 18 Dateien gespeichert. Das bedeutet, dass aufeinanderfolgende Dateien die Ergebnisse darstellen, die durch eine Drehung der Rotorblätter um 5° entstanden sind.

Ziel dieses Testfalls ist der Nachweis, dass einerseits mittels Analython mehrere Dateien nacheinander eingelesen und ausgewertet werden können. Andererseits sollen die Ergebnisse der Auswertung, die Kräfte und Momente, in einer Datei gespeichert werden können, um sie anschließend mittels dem Programm Tecplot (siehe hierzu Kapitel 5.2.2) ohne weitere Bearbeitung darzustellen.

Für die Testausführung ist die Skriptdatei zur Ausführung von Analython dementsprechend angepasst, dass darin das Einlesen und Auswerten der Ergebnisdateien der Strömungssimulation in einer so genannten Schleife, also nacheinander geschieht. Anschließend erfolgt die Ausgabe aller Kräfte und Momente dieser Auswertungen durch den Aufruf der Controller-Methode *writeForcesAndMomentaOfAllProfiles*.

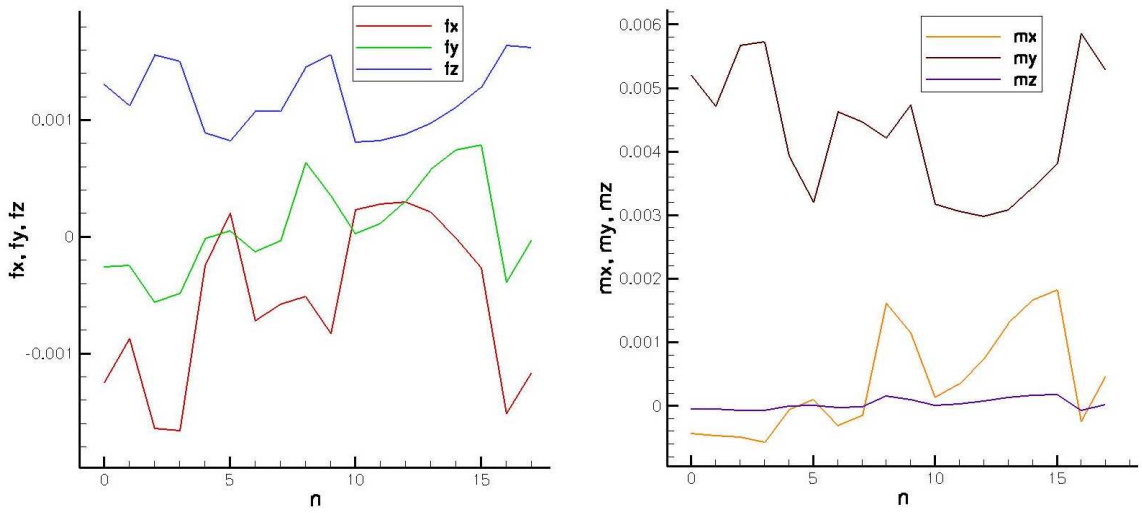


Abbildung 20: Ergebnis des DAUROT-Tests

Wie in Abbildung 20 erkennenbar ist, wurden die Kräfte (f_x, f_y, f_z) und Momente (m_x, m_y, m_z) für die 18 Eingabedateien, repräsentiert durch den Parameter n, be-

rechnet. Diese Daten wurden mittels Analython in einer Datei gespeichert und können dadurch mit Tecplot visualisiert werden. Die genauere Analyse der Ergebnisse ist in dieser Arbeit nicht erforderlich, da dies nicht zum Aufgabengebiet zählt.

7.5 Ergebnis

8 Bedienung des Programms

Für die Ausführung von Analython sind die Programme Python in der Version 2.2.3 (siehe Kapitel 4.2) und Numpy in der Version 23.1 (siehe Kapitel 5.2.1) erforderlich. Bei einem Einsatz anderer Versionen kann die korrekte Arbeitsweise von Anlython nicht garantiert werden, da ältere Versionen meist weniger Funktionsumfang bieten und neuere Versionen teilweise Funktionalitäten durch andere, verbesserte ersetzen.

Das Auswerteprogramm Analython kann in jedem beliebigen Verzeichnis gespeichert werden. Darin ist einerseits ein weiteres Verzeichnis namens *src* zu finden, indem die Quellcodedateien abgelegt sind. Andererseits existiert eine so genannte *README*-Datei, in der für den Benutzer viele Informationen zu Analython ersichtlich sind, die auch im Rahmen dieser Arbeit angesprochen wurden. Letztendlich liegt die Skript-Datei *analython.py* vor, mit der das Programm ausgeführt werden kann. Darin definiert der Benutzer die benötigten Variablen und kann anschließend die Methoden der Controller-Klasse aufrufen. Alle Methoden, die dem Benutzer zur Verfügung stehen, werden in diesem Kapitel einzeln aufgeführt und deren Einsatzmöglichkeiten erläutert. Die Parameter zur Übergabe an die Methode sind der Übersichtlichkeit wegen bewusst weggelassen, sie werden dennoch kurz beschrieben.

8.1 Eingabe, Speicherung, Ausgabe

Die grundlegensten Funktionalitäten des Programms Analython bestehen aus der Eingabe von Daten, deren Speicherung und Ausgabe. Dem Benutzer stehen hierfür die folgenden Methoden zur Verfügung.

1. readFileSaveData
   Mittels dieser Methode wird eine Datei eingelesen und deren Daten zwischengespeichert. Hierfür muss angegeben werden, ob es sich um eine strukturierte oder unstrukturiete Datei handelt. Des Weiteren ist der relative Pfad zur Datei anzugeben.

2. createNewProfile
   Mit dem zu übergebenden, eindeutigen Namen wird hiermit ein neues Profil angelegt.

3. addZonesToProfile

   Zu dem vorher angelegten Profil können einzelne oder alle Zonendaten der zuvor eingelesenen Dateien dem anzugebenden Profil hinzugefügt werden. Es ist somit beispielsweise möglich, ein Profil zu erstellen, dass aus den Daten mehrerer eingelesener Dateien besteht.

4. writeIntersection

   Die Daten einer zuvor ermittelten Schnittmenge können mit dieser Methode in einer Datei gespeichert werden. Hierfür ist anzugeben, um welche Schnittmenge es sich handelt, ob berechnete Kräfte und Momente mit gespeichert werden sollen und ein Pfad zur Datei, in der die Speicherung erfolgen soll.

5. writeProFile

   Mit dieser Methode werden die Daten eines Profils ähnlich zu 4 in einer Datei gespeichert.

6. writeDragAndLiftOfAllIntersections

   Diese Methode bewirkt, dass die Daten des Auftriebs und des Gesamtwiderstandes aller Schnittlinien in einer anzugebenden Datei gespeichert werden.

7. writeForcesAndMomentaOfAllProfiles

   Mittels dieser Methode besteht die Möglichkeit, die berechneten Werte der Druck-, Reibungs- und Gesamtkräfte sowie der Momente für die gesamten, gespeicherten Profile in einer Datei abzuspeichern.

8. deleteIntersection

   Zuvor angelegte Schnittmengen können mit dieser Methode wieder gelöscht werden, damit beispielsweise der bezeichnende Name der Schnittmenge nochmals genutzt werden kann.

9. deleteProfile

   Ein vorher angelegtes Profil kann hiermit gelöscht werden, damit zum Beispiel der Speicherplatz, der für die Daten benötigt wurde, wieder freigegeben wird.

8.2 Berechnung von Kräften und Momenten

10. calcDragAndLiftOfIntersection, calcDragAndLiftOfProfile

11. calcForcesOfIntersection, calcForcesOfProfile

12. calcMomentaOfIntersection, calcMomentaOfProfile

Mit den Methoden 10 bis 12 kann der Benutzer die Berechnungen von Kräften, Momenten, Auftrieb und Gesamtwiderstand ausführen. Dazu muss einzig angegeben werden, um welches Profil oder welche Schnittmenge es sich handelt.

13. setBladeAngleForIntersection, setBladeAngleForProfile
    Mittels dieser Methoden kann der Winkel angegeben werden, der für die Berechnung von Auftrieb und Widerstand (vergleiche Kapitel 3.3.4) notwendig ist.

14. setReferencePointForIntersection, setReferencePointForProfile
    Mit diesen Methoden lässt sich der Bezugspunkt angeben, der für die Berechnung der Momente (vergleiche Kapitel 3.3.5) erforderlich ist.

15. getTotalFrictionOfProfile, getTotalPressureOfProfile
    Hiermit hat der Benutzer die Möglichkeit, sich die gesamte Druck- sowie die gesamte Reibungskraft eines Profils auszugeben. Diese Möglichkeit ist speziell für spätere Weiterentwicklungen nötig.

8.3 Schnittlinienberechnung

Für die Ermittlung von Schnittlinien, die daurch entstehen, dass die Schnittmenge eines gegebenen Profils mit einer vorgegebenen Ebene berechnet wird, stehen die folgenden drei Methoden zur Verfügung.

16. cutProfileVia1Point2Vectors
    Mittels dieser Methode kann die Ebene in Parameterform angegeben werden. Das heißt, es müssen ein Punkt der Ebene und zwei in verschiedene Richtungen zeigende Richtungsvektoren übergeben werden.

17. cutProfileVia1PointNormalVector
    Diese Methode beschreibt die Ebene in Hessescher Normalform. Dazu müssen ein Punkt der Ebene und der Normalenvektor hierfür angegeben werden.

18. cutProfileVia3Points
    Mit drei Punkten, die innerhalb einer Ebene allerdings nicht alle auf einer Geraden liegen, lässt sich hierüber die Ebene definieren.

Für alle drei Methoden gilt, dass der jeweilige Name des Profils angegeben werden muss, das mit der Ebene geschnitten werden soll. Des Weiteren muss ein eindeutiger Name für die resultierende Schnittmenge übergeben werden. Bei doppelter Namensgebung bricht das Programm mit einer Fehlermeldung ab. Um dies zu vermeiden, besteht die Möglichkeit, eine nicht weiter benötigte Schnittmenge mit dem Befehl 8 zu löschen um anschließend den vorher genutzten Namen für eine Schnittmenge nochmals zu nutzen. Analython ermittelt aufgrund der Daten für alle Zonen die Schnittlinien und speichert diese in jeweils einer IntersectionLine-Instanz auf die die Intersection-Instanz mit benutzerdefiniertem Namen eine Referenz besitzt. Hierauf können die in Kapitel 8.2 aufgeführten Methoden angewendet werden.

8.4 Skalierung von Daten

Für die Skalierung sämtlicher Daten stehen dem Benutzer über die Controller-Klasse nachfolgende Methoden zur Verfügung.

19. scaleCoefficientsOfIntersection, scaleCoefficientsOfProfile
    Diese Methoden skalieren den Druck- und den Reibungsbeiwert.

20. scaleCoordinatesOfIntersection, scaleCoordinatesOfProfile
    Mittels dieser Methoden ist es möglich, die Koordinaten der Netzpunkte zu skalieren. Dies ist insbesondere für spätere Weiterentwicklungen notwendig, da sich hiermit Transformationen des Koordinatensystems durchführen lassen.

21. scaleDragAndLiftOfIntersection, scaleDragAndLiftOfProfile
    Der Gesamtwiderstand und der Auftrieb lassen sich mit diesen Methoden skalieren.

22. scaleForcesOfIntersection, scaleForcesOfProfile
    Sämtliche zuvor berechneten Kräfte (Druck-, Reibungs- und Gesamtkraft) kann der Benutzer mit diesen Methooden skalieren.

23. scaleMomentaOfIntersection, scaleMomentaOfProfile
    Die vorher ermittelten Momente können hiermit skaliert werden.

24. scaleSurfaceVectorsOfIntersection, scaleSurfaceVectorsOfProfile
    Für die Skalierung der Oberflächenvektoren stehen dem Benutzer diese Methoden zur Verfügung.

Die Methoden beziehen sich jeweils auf die Daten der angelegten Profile oder Schnittmengen. Das heißt, die Auswirkung ist die selbe, entweder auf die Profildaten oder die Daten der jeweiligen Schnittmenge. Für die Unterscheidung, auf welche Daten die Skalierungen durchgeführt werden sollen, existieren die Namenszusätze *Intersection* oder *Profile* für die Methoden. Demnach muss bei der entsprechenden Methode zusätzlich der Name des Profils oder der Schnittmenge übergeben werden.

Alle Skalierungsmethoden wurden aufgrund dessen implementiert, da im Institut für Aerodynamik und Strömungstechnik des DLR gelegentlich die Daten oder modifiziert werden müssen, weil sich Berechnungsvorschriften mit der Weiterentwicklung ändern.

9 Zusammenfassung

Im Rahmen dieser Diplomarbeit entstand das Programm Analython für die Auswertung von Daten der Strömungssimulation speziell für Hubschrauberkonfigurationen. Im Gegensatz zu früheren Programmen ist es in der Lage strukturierte und unstrukturierte Daten zu analysieren und Berechnungen darauf auszuführen. Die Datensätze können hierfür aus verschiedenen Dateien eingelesen und beliebig zusammengestellt werden. Die Berechnungen können auf einem gesamten Datensatz, dem so genannten Profil, oder auf einer Schnittmenge mit einer Ebene durchgeführt werden.
Eine Skriptdatei ermöglicht dem Benutzer die Reihenfolge der hohen Anzahl an Funktionalitäten vor dem Programmablauf beliebig festzulegen, um anschließend Analython automatisiert ablaufen zu lassen.

Es ist für den Benutzer als auch für den Entwickler ein sauber dokumentiertes, übersichtliches Programm entstanden. Dabei konnten weitere Erfahrungen im Rahmen der Objektorientierten Programmierung sowie der Programmiersprache Python gemacht werden. Des Weiteren gewährte diese Arbeit einen tieferen Einblick in die numerische Strömungssimulation.

Das Auswerteprogramm wird, anschließend an diese Arbeit, seinen Einsatz im institut für Aerodynamik und Strömungstechnik des DLR finden. Aufgrund der sich ständig ändernden Rahmenbedingungen, die durch den ständigen Fortschritt in der numerischen Simulation zu begründen sind, stellt Analython keine endgültige Version dar. Das Programm wurde auch im Hinblick darauf entwickelt, dass Weiterentwicklungen hieran einfach durchzuführen sind. So ist es beispielsweise möglich, Analython insofern zu erweitern, dass es in der Lage ist, Dateien mit anderer Struktur oder abweichenden Daten einzulesen. Ebenso können problemlos weiterführende Berechnungen integriert werden, beziehungsweise die Ausgabe der Auswertung entsprechend angepasst werden. Ein hilfreiches Werkzeug für die Ingenieure der Strömungssimulation könnte eine GUI[5] sein, insofern diese sich nicht mit der Programmierung auskennen. Diese GUI dient dazu, die Modifikationen nicht in der Skriptdatei durchführen zu müssen. Statt dessen kann der Benutzer den Programmablauf sich auf grafischer Ebene zurechtklicken, um sich anschließend mit der GUI die Skriptdatei erstellen zu lassen.

[5] Graphical User Interface, grafische Benutzerschnittstelle

Abbildungsverzeichnis

Literatur

[1] Gerhard Merziger, Günter Mühlbach, Detlef Wille, Thomas Wirth:
Formeln + Hilfen zur höheren Mathematik,
Binomi Verlag, Stand Oktober 2001

[2] Allen Downey, Jeffrey Elkner, Chris Meyers:
How to Think Like a Computer Scientist - Learning with Python,
Quality Books, Inc., Stand April 2002

[3] VR Kreditwerk Hamburg - Schwäbisch Hall AG: *Jumli* Internetseite
http://www.jumli.de/, Stand August 2005

[4] Herbert Oertel jr., Eckart Laurien: *Numerische Stömungsmechanik*,
Vieweg Verlag, Januar 2003

[5] OSTG Open Source Technology Group: *Numpy* Download Internetseite
http://sourceforge.net/projects/numpy/, Stand Juli 2005

[6] Python Software Foundation: *Python* Download Internetseite,
http://www.python.org/download/, Stand Juli 2005

[7] Uwe Schneider, Dieter Werner: *Taschenbuch der Informatik*
Fachbuchverlag Leipzig, Stand 2001

[8] Tecplot, Inc.: *Tecplot User's Manual Version 10*
Stand März 2005

[9] Jochen Wild: *Thrust/Drag Bookkeeping and Aerodynamic*
Force Breakdown over Components,
DLR-Institutsbericht, Stand 28 Juni 1999

[10] Prof. Dr.-Ing. R. Radespiel: Skript zur Vorlesung *Tragflügelaerodynamik*,
Technische Universität Braunschweig, Ausgabe WS 2004/2005

[11] Internetlexikon: *WIKIPEDIA, Die freie Enzyklopädie*,
http://de.wikipedia.org, Stand September 2005