Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Pradeep Subedi

Berechnung und thermodynamische Bewertung einer Gasturbine in der Python Programmiersprache und Vergleich der Vorgehensweise und der Ergebnisse mit einer vorhandenen Matlab-Modellierung

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Pradeep Subedi

Berechnung und thermodynamische Bewertung einer Gasturbine in der Python Programmiersprache und Vergleich der Vorgehensweise und der Ergebnisse mit einer vorhandenen MATLAB-Modellierung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

Im Studiengang Energie- und Anlagensysteme
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer/in: Dr. Ing. Thomas Flower
Zweitprüfer/in: Prof. Dr.-Ing. Achim Schmidt

Abgabedatum: 23.04.2019

I. Zusammenfassung

Pradeep Subedi

Thema der Bachelorarbeit

Berechnung und thermodynamische Bewertung einer Gasturbine in der Python Programmiersprache und Vergleich der Vorgehensweise und der Ergebnisse mit einer vorhandenen MATLAB-Modellierung.

Stichworte

Massen- und Stoffmengenkonzentration, Eintritts- und Austrittstemperatur, Druck, Enthalpie, Entropie, Exergie, Energie, Heizwert, Wirkungsgrad.

Kurzzusammenfassung

Diese Arbeit bezieht sich auf die systematische und strukturierte Simulation einer Gasturbine in der Python Programmiersprache. Nach kurzer Einführung in der Python Programmiersprache werden an jeder Komponente der Gasturbine thermodynamischen Eigenschaften der Fluide berechnet sowie die Energie bilanziert. Abschließend werden die Vorgehensweise und die Ergebnisse mit der vorhandenen MATLAB-Berechung verglichen.

Pradeep Subedi

Bachelor Thesis title

Thermodynamic evaluation of a gas turbine in python programming language and comparison of the procedure and the results with the existing MATLAB model

Keywords

Mass and Mole concentration, inlet and outlet temperature, pressure, enthalpy, entropy, exergy, energy, lower heating value, efficiency.

Abstract

The thesis relates to the systematic and structural simulation of a gas turbine in python programming language. After a small introduction in the python programming langaue, thermodynamic properties of the gases will be calculated, and energy will be balanced. Finally, the python procedure and results of the calculation will be compared with the MATLAB one demonstrating the difference between the each method.

II. Inhaltsverzeichnis

III. Abbildungsverzeichnis

IV. Tabellenverzeichnis

V. Formelverzeichnis

Symbol	Einheit	Beschreibung
$\dot{m}_L$	[kg/s]	Luftmassenstrom
$\dot{m}_B$	[kg/s]	Brennstoffmassenstrom
h_1	[kJ/kg]	Absolute spezifische Enthalpie vor der isentropen Verdichtung
T_1	[K]	Temperatur vor der isentropen Verdichtung
p_1	[bar]	Druck vor der isentropen Verdichtung
h_2	[kJ/kg]	Absolute spezifische Enthalpie nach der isentropen Verdichtung
T_2	[K]	Temperatur nach der isentropen Verdichtung
p_2	[bar]	Druck nach der isentropen Verdichtung
h_3	[kJ/kg]	Absolute spezifische Enthalpie nach der Verbrennung
T_3	[K]	Temperatur nach der Verbrennung
p_3	[bar]	Druck nach der Verbrennung
h_4	[kJ/kg]	Absolute spezifische Enthalpie nach der isentropen Entspannung
T_4	[K]	Temperatur nach der isentropen Entspannung
p_4	[bar]	Druck nach der isentropen Entspannung
c_p	[kJ/kgK]	Absolute spezifische Wärmekapazität
κ	[-]	Isentroper Exponent
$\dot{W}_V$	[W]	Verdichter Leistung
$\dot{Q}_{zu}$	[W]	Wärmezufuhr bei konstantem Druck
$\dot{W}_T$	[W]	Turbinenleistung

P_{gesamt}	[W]	Nettoleistung des Joule-Prozesses
η_{th}	[-]	Thermische Gesamtwirkungsgrad des Joule-Prozesses
tmp_ref	[K]	Niedrigere Temperatur für die durchschnittliche spezifische Wärmekapazität
e_m	[kJ/kmol]	Absolute molare spezifische Exergie
e	[kJ/kg]	Absolute spezifische Exergie
h_m	[kJ/kmol]	Absolute spezifische molare Enthalpie
h	[kJ/kg]	Absolute spezifische Enthalpie
$h_{mUmgebungsluft}$	[kJ/kmol]	Absolute spezifische molare Enthalpie der Luft beim Umgebungszustand
$h_{Umgebungsluft}$	[kJ/kg]	Absolute spezifische Enthalpie der Luft beim Umgebungszustand
$T_{Umgebungsluft}$	[K]	Umgebung Temperatur der Luft
s_m	[kJ/kmolK]	Absolute spezifische molare Entropie
s	[kJ/kgK]	Absolute spezifische Entropie
$s_{mUmgebungsluft}$	[kJ/kmolK]	Absolute spezifische molare Entropie der Luft beim Umgebungszustand
$s_{Umgebungsluft}$	[kJ/kgK]	Absolute spezifische Entropie der Luft beim Umgebungszustand
$T_{2,isen,V}$	[K]	Temperatur nach der isentropen Verdichtung
$h_{2,m}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der realen Verdichtung
$h_{1,m}$	[kJ/kmol]	Absolute spezifische molare Enthalpie vor der realen Verdichtung
$h_{2,m,isen}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der isentropen Verdichtung
$\eta_{V,isen}$	[-]	Wirkungsgrad des Verdichters
Hu_m	[kJ/kmol]	Spezifischer molarer Heizwert

Hu	[kJ/kg]	Spezifischer Heizwert
$\dot{n}_L$	[kmol/s]	Stoffmengenstrom der Luft
$\dot{m}_L$	[kg/s]	Massenstrom der Luft
$h_{m,L}$	[kJ/kmol]	Absolute spezifische molare Enthalpie der Luft
h_L	[kJ/kg]	Absolute spezifische Enthalpie der Luft
$\dot{n}_B$	[kmol/s]	Stoffmengenstrom des Brennstoffs
$\dot{m}_B$	[kg/s]	Massenstrom des Brennstoffs
$h_{m,B}$	[kJ/kmol]	Absolute spezifische molare Enthalpie des Brennstoffs
h_B	[kJ/kg]	Absolute spezifische Enthalpie des Brennstoffs
$\dot{n}_{Abgas}$	[kmol/s]	Stoffmengenstrom des Abgases
$\dot{m}_{Abgas}$	[kg/s]	Massenstrom des Abgases
h_{mAbgas}	[kJ/kmol]	Absolute spezifische molare Enthalpie des Abgases
h_{Abgas}	[kJ/kg]	Absolute spezifische Enthalpie des Abgases
$h_{m,B,Umgebungszustand}$	[kJ/kmol]	Absolute spezifische molare Enthalpie des Brennstoffs beim Umgebungszustand
$h_{B,Umgebungszustand}$	[kJ/kg]	Absolute spezifische Enthalpie des Brennstoffs beim Umgebungszustand
$s_{m,B}$	[kJ/kmolK]	Absolute spezifische molare Entropie des Brennstoffs
s_B	[kJ/kgK]	Absolute spezifische Entropie des Brennstoffs
$s_{m,B,Umgebungszustand}$	[kJ/kmolK]	Absolute spezifische molare Entropie des Brennstoffs beim Umgebungszustand
$s_{B,Umgebungszustand}$	[kJ/kgK]	Absolute spezifische Entropie des Brennstoffs beim Umgebungszustand
L_{min}	[-]	Mindestluftbedarf (bezogen auf die Stoffmenge des Brennstoffs)
$x_{O_2,tr.\ Luft}$	[-]	Stoffkonzentration von Sauerstoff in der Luft

O_{min}	[-]	Mindestsauerstoffbedarf
$\dot{n}_{L,min}$	[kmol/s]	Mindeststoffmengenstrom der Luft
λ	[-]	Verbrennungsluftverhältnis
$\dot{n}_B$	[kmol/s]	Brennstoffstoffmengenstrom
$h_{3,m}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der Verbrennung
$\dot{n}_2$	[kmol/s]	Stoffmengenstrom der Luft
$\dot{n}_9$	[kmol/s]	Stoffmengenstrom des Brennstoffes
$\dot{n}_3$	[kmol/s]	Stoffmengenstrom des Abgases
$h_{9,m}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der Brennstoffvorwärmung
$\dot{Q}_v$	[%]	Thermischer Verlust der Brennkammer
$T_{steigung}$	[K]	Lineare Steigung der Temperatur zwischen λ_1 und λ_6
$T_{schätzwert}$	[K]	Temperatur Schätzwert nach der Verbrennung
$\lambda_{Schätzwert}$	[-]	Verbrennungsluftverhältnis Schätzwert nach der Verbrennung
λ_6	[-]	Verbrennungsluftverhältnis = 6
λ_1	[-]	Verbrennungsluftverhältnis = 1
$T_{\lambda=6}$	[K]	Temperatur nach der Verbrennung für λ_6
$T_{\lambda=1}$	[K]	Temperatur nach der Verbrennung für λ_1
$h_{4,m}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der realen Entspannung
$h_{4,m,isen}$	[kJ/kmol]	Absolute spezifische molare Enthalpie nach der isentropen Entspannung
$\eta_{T,isen}$	[-]	Wirkungsgrad der Turbine

1 Einleitung

1.1 Einführung

Mithilfe der Gasturbine wird die thermische Energie in mechanische Energie umgewandelt. Diese mechanische Energie wiederum nutzt die Gasturbine, um einen Generator anzutreiben und elektrische Energie zu erzeugen. Die ersten Pläne für ein Gasturbine wurde im Jahr 1791 von dem Engländer John Barber patentiert [1]. In den Jahren 1905/06 wurde eines der ersten stromproduzierenden Gasturbinenwerke von etwa 4 bis 6 kW und einem Wirkungsgrad zwischen 2 und 3 % gebaut.[1] Nach dem Zweiten Weltkrieg wurden Gasturbinen prinzipiell für die Flugtriebwerksauslegung entwickelt. Erst in den letzten Jahrzehnten konnten die thermischen Wirkungsgrade der stationären Gasturbinen auf 30 % im offenen Prozess gesteigert werden [2]. Die stationären Gasturbinen werden heutzutage in den meisten Fällen in Kombination mit der Dampfturbine eingesetzt, um einen höheren Gesamtwirkungsgrad der Gas- und Dampfturbinenanlage zu erreichen.

Zur thermodynamischen Berechnung einer Gasturbine sind zahlreiche Berechnungstools verfügbar. In dieser Arbeit wird die Gasturbine in der Programmiersprache Python simuliert.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, eine alternative, kostengünstige Simulationsmethode zur thermodynamischen Berechnung der Gasturbine darzustellen. Hierzu wird eine Gasturbinenanlage in der Python-Programmiersprache simuliert und die Vorgehensweise bzw. die Ergebnisse werden mit der MATLAB-Modellierung verglichen. Als Vergleichsgrößen werden sowohl die thermodynamischen Kenngrößen an jedem Punkt der Gasturbine als auch die energetische und exergetische Bilanzierung von jedem Teil herangezogen. Des Weiteren werden die Zusammensetzung des Rauchgases nach der Verbrennung sowie der Heizwert und der Wirkungsgrad der Gasturbine ermittelt.

[1] Christof Lechner, Jörg Seume, Hrsg.: Stationäre Gasturbinen, S. 4.

2 Theoretische Grundlagen

2.1 Aufbau und Wirkungsweise von Gasturbinen

Die Gasturbine besteht prinzipiell aus einem Lufteintritt, einem Luftverdichter, einer Verbrennungskammer, einer Turbine, einem Abgasauslass, einer Welle von der Turbine zum Verdichter, einem Generator mit einer Abtriebswelle für Wellentriebwerke und einem Abgasauslass (siehe Abbildung 2.1-1).

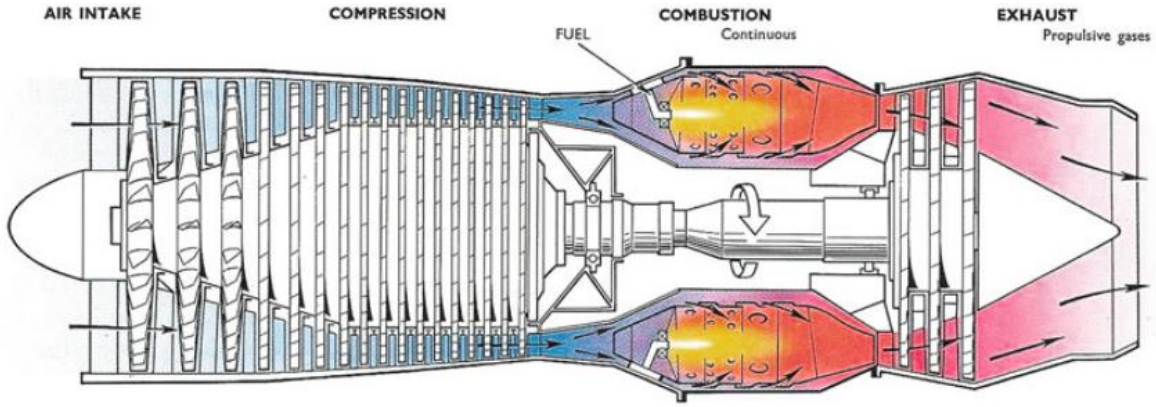

Abbildung 2.1-1: Gasturbinenanlage [3]

Grundsätzlich wird zwischen offenen und geschlossenen Gasturbinenanlagen unterschieden. Bei der offenen Gasturbinenanlage erfolgt die Wärmezufuhr in der Brennkammer im Arbeitsfluid. Die Verbrennungsgase werden nach der Expansion in der Turbine in die Umgebung freigesetzt. Die geschlossene Gasturbinenanlage verfügt anstatt der Brennkammer über Wärmetauscher, in denen die Wärme dem Prozess zugeführt wird. Die Wärmeabfuhr erfolgt ebenfalls in einem Wärmetauscher, so dass das Arbeitsfluid in einem geschlossenen System umläuft [5].

In dieser Arbeit wird eine offene Gasturbinenanlage dargestellt. Abbildung 2.1-2 zeigt eine schematische Darstellung der offenen Gasturbinenanlage. Der Lufteintritt dient der strömungsdynamischen Anpassung zwischen der Einsatzumgebung und dem Verdichter. Hier wird die Luft verunreinigt und es erfolgt ein Druckverlust. Nach dem Lufteinlauf strömt die Luft in den Verdichter. Dieser hat die Aufgabe, die angesaugte Luft auf einen möglichst hohen Druck zu komprimieren, um einen besseren

Gasturbinenwirkungsgrad zu erreichen. Ein Teil der Brennstoffenergie wird zum Antrieb des Verdichters benötigt deshalb ist ein guter Verdichterwirkungsgrad notwendig, um einen hohen thermischen Wirkungsgrad der Gasturbine zu erzielen. Nach dem Verdichter tritt die Luft mit dem hohen Druck und der hohen Temperatur in die Brennkammer ein. Dort wird der Brennstoff bei annähernd konstantem Druck verbrannt. Dabei heizt sich das Abgas auf Temperaturen von 1500 °C und mehr auf[2]. Vor der Verbrennung wird der Brennstoff in einem Kompressor und einem Vorwärmer vorbereitet. Dabei werden der Druck und die Temperatur des Brennstoffes erhöht und für die Verbrennung angepasst. Die heißen Verbrennungsgase aus der Brennkammer strömen durch die Turbine und werden dabei entspannt, d. h., sowohl Druck als auch Temperatur sinken [6]. Hierbei gibt das energiereiche, heiße Abgas die Leistung während des Expansionsvorganges an die Turbine ab. Ein Drittel dieser Leistung steht als Nutzleistung, z. B. zum Antrieb eines Generators bzw. als Kompressor- oder Pumpenantrieb, zur Verfügung und etwa zwei Drittel werden zum Antrieb des Verdichters benötigt [6]. Nach der Entspannung in der Turbine auf Umgebungsdruck besitzt das Abgas noch eine hohe Temperatur (ca. 500 °C)[2].

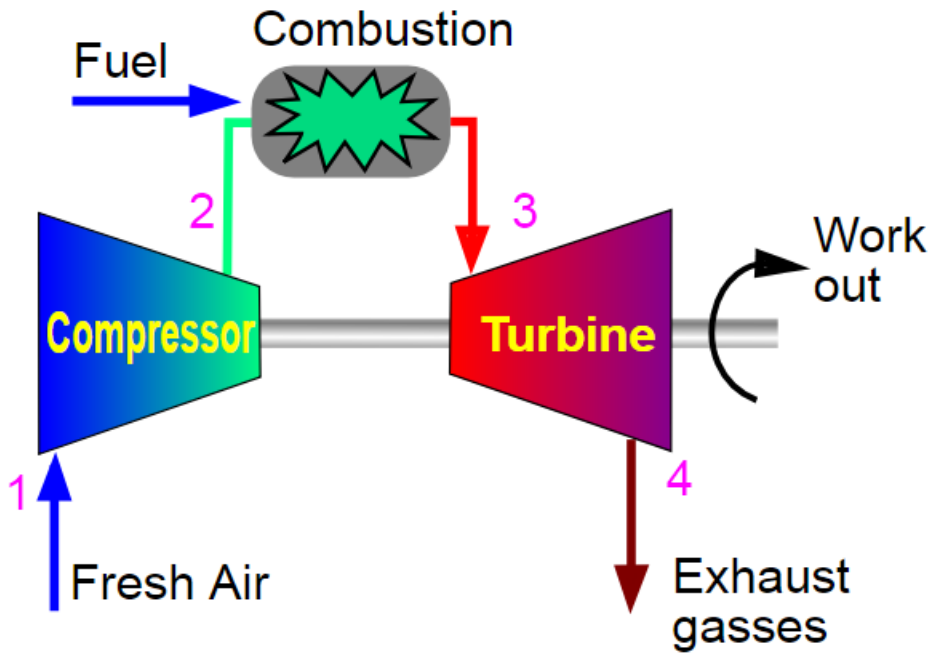

Abbildung 2.1-2: Schematische Darstellung der Gasturbine mit offenem Fließprozess [4]

[2] https://www.asue.de/sites/default/files/asue/themen/gasturbinen/1994/broschueren/11_09_94_1994_grundlagen_technik_betrieb.pdf

2.2 Der ideale Kreisprozess

Der rechtslaufende Joule-Kreisprozess ist ein idealer Vergleichsprozess für den in Gasturbinen ablaufenden Vorgang und wird unter folgenden Voraussetzungen herangezogen:

- Isentrope Verdichtung
- Isobare Verbrennung
- Isentrope Expansion
- Isobare Wärmezufuhr

Der ideale Kreisprozess ist schematisch im p-v-Diagramm (siehe Abbildung 2.2-1) und T-s-Diagramm (siehe Abbildung 2.2-2) aufgezeichnet.

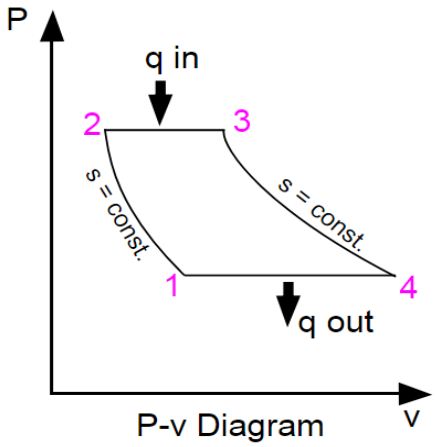

Abbildung 2.2-1: p-v-Diagramm idealer Joule-Kreisprozess [4]

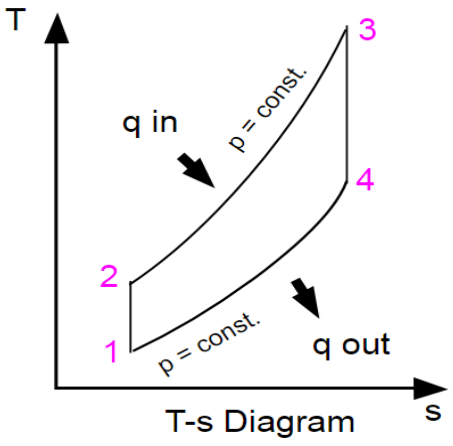

Abbildung 2.2-2: T-s-Diagramm idealer Joule-Kreisprozess [4]

Der erste Prozess des Joule-Zyklus ist eine adiabate isentrope Kompression von 1 nach 2. Da die Kompression adiabatisch ist, erfolgt keine Wärmeübertragung und die zum Betrieb des Kompressors erforderliche Leistung ist:

$$\dot{W}_{12} = \dot{m}_L \cdot (h_2 - h_1) \tag{1}$$

Es handelt sich hier um ein ideales Gas und somit können Enthalpiedifferenzen durch Temperaturdifferenzen ersetzt werden. Außerdem wird angenommen, dass die spezifische Wärmekapazität c_p während des Kreisprozesses konstant bleibt.

$$\dot{W}_V = \dot{m}_L \cdot c_p \cdot (T_2 - T_1) \tag{2}$$

Der zweite Prozess ist eine Wärmezufuhr bei konstantem Druck:

$$\dot{Q}_{zu} = (\dot{m}_L + \dot{m}_B) \cdot h_3 - \dot{m}_L \cdot h_2 \tag{3}$$

Es wird angenommen, dass der Brennstoffmassenstrom im Vergleich zu dem vom Verdichter angesaugten Luftmassenstrom vernachlässigbar klein $\dot{m}_B \ll \dot{m}_L$ ist [18]. Es folgt:

$$\dot{Q}_{zu} = \dot{m}_L \cdot c_p \cdot (T_3 - T_2) \tag{4}$$

Vom Zustand 3 nach 4 folgt die isentrope Entspannung des Abgases. Die Turbinenleistung berechnet sich wie folgt:

$$\dot{W}_T = (\dot{m}_L + \dot{m}_B) \cdot (h_3 - h_4) \tag{5}$$

Mit $\dot{m}_B \ll \dot{m}_L$ und den Annahmen aus der Gleichung (2) ergibt sich:

$$\dot{W}_T = \dot{m}_L \cdot c_p \cdot (T_3 - T_4) \tag{6}$$

Die Gesamtleistung des Joule-Prozesses ergibt sich somit aus der Differenz von Turbinen- und Verdichterleistung:

$$P_{gesamt} = \dot{W}_T - \dot{W}_V \tag{7}$$

Der thermische Gesamtwirkungsgrad einer Gasturbine ergibt sich somit als Quotient aus Nutzleistung (7) und zugeführter Wärme (4):

$$\eta_{th} = \frac{Nutzen}{Aufwand} = \frac{P_{gesamt}}{\dot{Q}_{zu}}$$

$$= \frac{\dot{m}_L \cdot c_p \cdot (T_3 - T_4) - \dot{m}_L \cdot c_p \cdot (T_2 - T_1)}{\dot{m}_L \cdot c_p \cdot (T_3 - T_2)}$$

$$= \frac{(T_3 - T_2) - (T_4 - T_1)}{(T_3 - T_2)}$$

$$= 1 - \frac{(T_4 - T_1)}{(T_3 - T_2)} \tag{8}$$

Für die isentrope Verdichtung bzw. Expansion gilt folgende Beziehung zwischen Druck und Temperatur bei konstantem Isentropenexponenten κ:

$$\frac{T_2}{T_1} = \left(\frac{p_2}{p_1}\right)^{\frac{\kappa-1}{\kappa}} \text{ bzw. } \frac{T_3}{T_4} = \left(\frac{p_3}{p_4}\right)^{\frac{\kappa-1}{\kappa}} \tag{9}$$

Die Druckverluste werden bei den Zustandsänderungen (im Verdichter, in der Brennkammer und in der Turbine) vernachlässigt. Somit erfolgt das Druckverhältnis bei isentroper Zustandsänderung in Verdichter und Turbine gleich:

$$\frac{p_2}{p_1} = \frac{p_3}{p_4} \tag{10}$$

Aus (9) und (10) ergibt sich:

$$\frac{T_2}{T_1} = \frac{T_3}{T_4} \tag{11}$$

Durch Kombination von (8), (9) und (11) lässt sich der thermische Wirkungsgrad wie folgt darstellen:

$$\eta_{th} = 1 - \frac{T_1 \cdot \left(\frac{T_4}{T_1} - 1\right)}{T_2 \cdot \left(\frac{T_3}{T_2} - 1\right)} = 1 - \frac{T_1}{T_2} \tag{12}$$

$$\eta_{th} = 1 - \frac{T_4 \cdot \left(1 - \frac{T_1}{T_4}\right)}{T_3 \cdot \left(1 - \frac{T_2}{T_3}\right)} = 1 - \frac{T_4}{T_3} \tag{13}$$

Der Wirkungsgrad in Abhängigkeit vom Druckverhältnis lässt sich dann wie in der Gleichung (14) beschreiben:

$$\eta_{th} = 1 - \frac{1}{\left(\frac{p_2}{p_1}\right)^{\frac{\kappa-1}{\kappa}}} \quad \text{bzw.} \quad \eta_{th} = 1 - \frac{1}{\left(\frac{p_3}{p_4}\right)^{\frac{\kappa-1}{\kappa}}} \tag{14}$$

Der Wirkungsgrad einer einfachen Gasturbine liegt zwischen 42 % und 45 %. Das Erhöhen des Druckverhältnisses am Kompressor ist der direkteste Weg, um den gesamten thermischen Wirkungsgrad eines Joule-Kreisprozesses zu erhöhen, da der thermodynamische Wirkungsgrad hauptsächlich vom Druckverhältnis $\left(\frac{p_2}{p_1}\right)$ abhängt. Je höher zudem die Gaseintrittstemperatur in der Turbine ist, desto höher ist die Turbinenleistung und somit auch die höhere Nutzleistung. Durch Erhöhung der Turbineneintrittstemperatur können sowohl die Nutzleistung als auch der thermische Wirkungsgrad verbessert werden. Es ist aber darauf zu achten, dass die Turbineneintrittstemperatur die Temperaturtragfähigkeit (1200 °C bis 1400 °C) des Materials der Turbine nicht übersteigt. Neben diesen beiden Faktoren ist von Bedeutung, dass die einzelnen

Komponenten wie der Verdichter, die Verbrennungskammer und die Turbine hoch effizient sind, um eine bessere Gasturbinenleistung zu erreichen.

3 Einführung in Python

Der Name dieser Programmiersprache hat nichts mit der Schlange zu tun, sondern mit der britischen Komödie-Serie Monty Python's Flying Circus. Anfang der 1990er Jahre wurde die Sprache von dem niederländischen Softwareentwickler Guido van Rossum, der Fan von Monty Python's Flying Circus war, am Centrum Wiskunde & Informatica in Amsterdam als Nachfolgerin der Programmier-Lehrsprache ABC entwickelt [6]. Im Jahr 1994 wurde die erste Vollversion von Python unter der Bezeichnung Python 1.0. veröffentlicht. Seitdem wurden mehrere robuste und verbesserte Versionen entwickelt. Die neueste Version gibt es seit dem 27. Juni 2018 unter der Bezeichnung Python 3.7.

Python verfügt über effiziente und abstrakte Datenstrukturen und einen einfachen, aber effektiven Ansatz zur objektorientierten Programmierung. Es beinhaltet zudem zahlreiche kostenlose Bibliotheken und dies ermöglicht es, die Python-Programmiersprache für fast jede Anwendung zu nutzen. Einige Bereiche, in denen mit Python gearbeitet wird, sind:

- Bei der Erstellung dynamischer Webseiten und der Entwicklung von Desktop-Anwendungen [7].
- Data-Science, maschinelles Lernen und künstliche Intelligenz [7]
- Software- und Spielentwicklung [7]
- Wissenschaftliche und numerische Berechnungen und Simulationen [7]
- Wirtschaftlichkeitsanalyse und E-Commerce-Systeme [7]

Mit Python können Programme mit grafischen Oberflächen nicht direkt geschrieben werden, dennoch ist eine Umsetzung problemlos möglich, weil viele kostenlose grafische Bibliotheken zur Verfügung stehen.

Es gibt zudem zahlreiche kostenlose Webseiten und Kurse, mit deren Hilfe Python online erlernt werden kann. Empfehlenswert sind die vier aufeinander aufbauenden Lernkurse *Computing in Python* der Technischen Hochschule Georgia auf der Webseite www.edx.org. Die für die diese Bachelorarbeit benötigten Python-

Programmierkenntnisse hat der Verfasser durch diese Kurse erlernt. Außerdem sind auf den Seiten www.coursera.org, www.udacity.com und www.udemy.com weitere kostenlose Einführungskurse zu Python zu finden.

3.1 Python Installation

Auf der offiziellen Python-Webseite www.python.org lässt sich je nach Betriebssystem, die neueste Version Python 3.7 herunterladen. Eine alternative Installationsmöglichkeit ist die Installation mit Anaconda unter www.anaconda.com. Anaconda ist eine *Freemium Open Source Distribution* für die Programmiersprachen Python und R, die unter anderem die Entwicklungsumgebung SPYDER, den Kommandozeilninterpreter *IPython* und ein webbasiertes Frontend für *Jupyter* enthält [8]. Mit Anaconda lassen sich auch viele grundlegende Bibliotheken für Python automatisch installieren, deshalb ist diese Installationsmethode zu empfehlen. Außerdem hat Anaconda eine eigene Kommandozeile, mit deren Hilfe die Python-Bibliotheken heruntergeladen und der Editor und die Python-Version aktualisiert werden können.

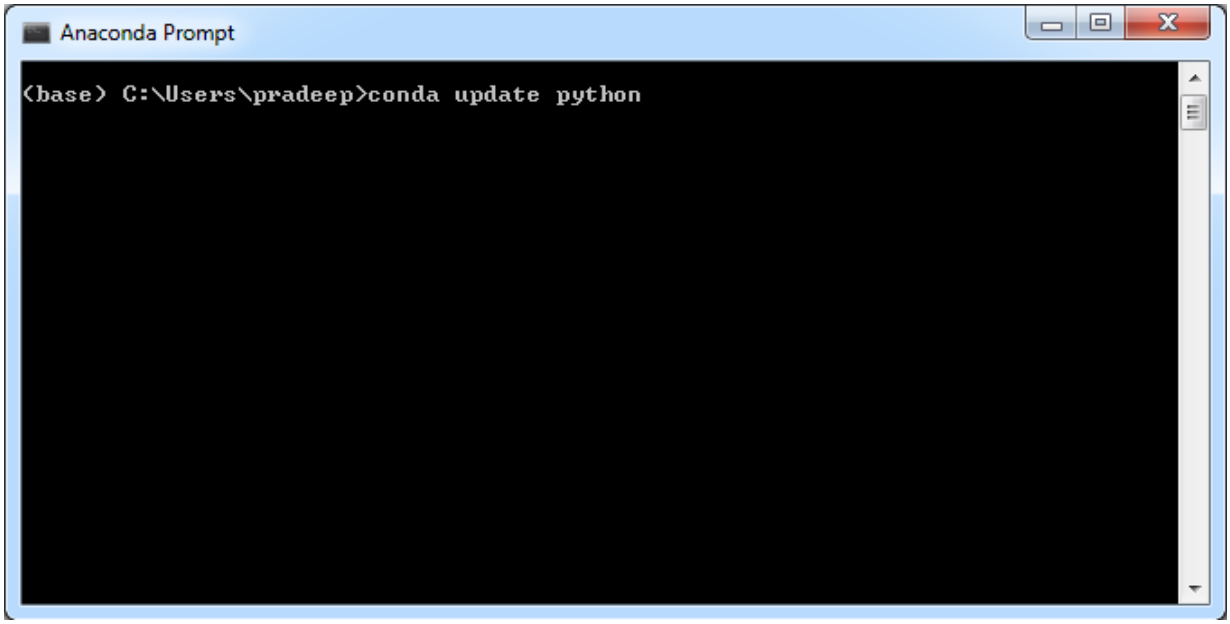

Abbildung 3.1-1: Anaconda Kommandozeile

3.2 Auswahl des Editors

Je nach Anwendung gibt es zahlreiche Python-Editoren und -Konsolen, in denen man Python-Skripte schreiben bzw. ausführen kann. Für die technische und wissenschaftliche Analyse wird am häufigsten SPYDER verwendet. SPYDER wurde im Jahr 2009 von Pierre Raybaut für die wissenschaftliche Anwendung konzipiert [9]. Es ist eine interaktive Entwicklungsumgebung für die Python-Sprache mit erweiterten Bearbeitungs-, Analyse und Debugging-Funktionen. Das Berechnungstool SPYDER steht für *Scientific Python Development Environment* [10]. Mit der Installation von Python über Anaconda wird SPYDER automatisch mitinstalliert.

Um SPYDER auf die neueste Version zu aktualisieren, wird der Befehl „*conda update Spyder*“ in die Kommandozeile von Anaconda eingegeben (siehe Abb. 3.1.1).

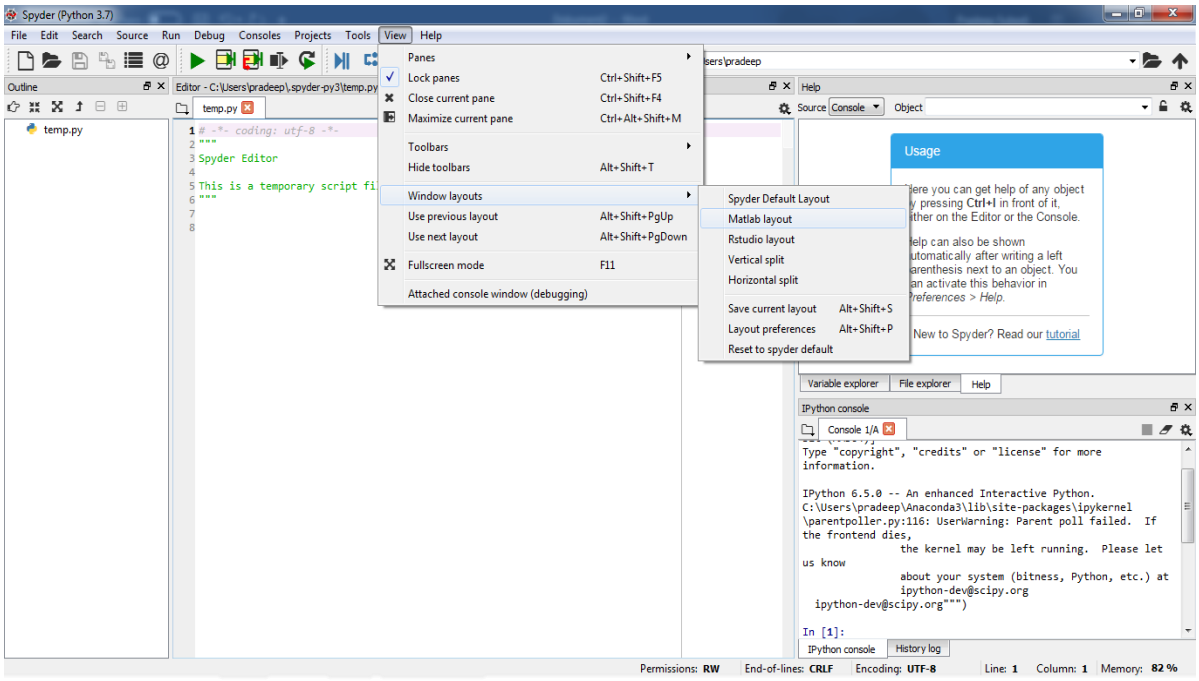

Abbildung 3.2-1: SPYDER Editor

Für MATLAB-Benutzer bietet SPYDER auch die Möglichkeit mit dem MATLAB Bildschirmentwurf zu arbeiten (siehe Abb. 3.2-1).

3.3 Bibliotheken

Da Python nur wenige Funktionen zur Verfügung hat, werden für alle Anwendungen die Bibliotheken benötigt. Für die Berechnung der Gasturbine wurden hauptsächlich vier Python-Bibliotheken benutzt.

- **NumPy**

  NumPy ist ein Akronym für ‚Numerisches Python' (englisch: ‚Numerical Python') [11]. Dabei handelt es sich um eine Bibliothek für Python, die Unterstützung für große mehrdimensionale Arrays und Matrizen sowie eine große Sammlung höherer mathematischer Funktionen für die Bearbeitung dieser Arrays bereitstellt. NumPy lässt sich in Python mit dem Befehl „*conda install NumPy*" in der Anaconda-Kommandozeile installieren. Nach der Installation muss die Bibliothek NumPy in das Python-Skript importiert werden, um deren Funktionen verwenden zu können. NumPy verfügt über zwei Arten von Arrays: 1D-Arrays und 2D-Arrays.

  Beispiel 1D-Array:

```
01. import numpy
02. A = numpy.array([1, 2, 3])
```

  Das Array mit der Bezeichnung A ist ein eindimensionales Array mit der Dimension (3). Die Dimension eines NumPy-Arrays lässt sich mit

```
01. A.shape
```

  bestimmen.

  Beispiel 2D-Array:

```
01. import numpy
02. B =numpy.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

  Das Array mit der Bezeichnung B ist ein dreidimensionales Array mit der Dimension (3,3).

  Bei der Berechnung der Gasturbine wurden größere Menge von Daten sowie die Werte, die sich während der Simulation ändern als NumPy Arrays definiert.

- **Pandas**

  Das Wort Pandas ist eine Abkürzung für ‚Panel Data'. Pandas ist eine leistungsstarke Open-Source-Bibliothek für die Datenanalyse in Python, die 2008 von Wes McKinney entwickelt wurde [12]. Pandas zielt auf mehrere typischen Schritte im Zuge der Verarbeitung und Untersuchung von Daten, unabhängig davon, woher die Daten stammen. Der erste Schritt besteht darin, die Daten einzulesen. Besonders aufgrund der Vielzahl bestehender Formate sparen die von Pandas gelieferten Werkzeuge Zeit. Die Bibliothek liest Datensätze in den Formaten CSV, Excel, HDF, SQL, JSON und HTML ein. Danach folgen die Vorbereitung und Manipulierung der eingelesenen Daten. Je nach Anwendung werden die manipulierten Daten im originalen Format zurückgegeben oder in einem anderen Format gespeichert. Pandas lässt sich in Python mit dem Befehl „*conda install pandas*" in der Anaconda-Kommandozeile installieren. Zur Berechnung der Gasturbine wurde Pandas lediglich verwendet, um die Einsatzdaten aus der Excel-Datei einzulesen. Die eingelesenen Daten speichert Pandas als Dataframe. Für die mathematische Bearbeitung von Arrays bietet NumPy eine effizientere Möglichkeit. Deshalb wurden die Dataframes in NumPy-Arrays umgewandelt.

  Nach der Installation muss die Pandas-Bibliothek, genauso wie NumPy, in das Python-Skript importiert werden, damit deren Funktionen verwendet werden können.

- **IAPWS**

  Die IAPWS ist eine im Jahr 1929 gegründete internationale Union von zwölf Organisationen, die sich der Untersuchung der Eigenschaften von Wasser und Wasserdampf gewidmet haben. Die Abkürzung steht für *International Association for the Properties of Water and Steam* [13]. Das Hauptziel der IAPWS ist es, weltweit anerkannte Formulierungen für die thermodynamischen Eigenschaften von gesättigtem und ungesättigtem Dampf, Wasser und wässrigen Lösungen für wissenschaftliche und industrielle Einsätze zur Verfügung zu stellen. Die Bibliothek der Python-Programmiersprache, um die thermodynamischen Eigenschaften von Wasser und Dampf zu ermitteln, ist *iapws*. Bei der

Gasturbinenberechnung wird mithilfe der Funktion *iapws97* aus der Bibliothek *iapws* der Sättigungsdruck von Wasser in Ansaugluft berechnet. Die Iapws97 ist eine im Jahr 1997 neue formulierte Approximation zur Berechnung der thermodnamischen Eigenschaften vom Dampf und Wasser [14]. *Iapws* muss wie andere Bibliotheken mit dem Befehl „*conda install iapws“* in der Anaconda-Kommandozeile installiert werden. Um diese Bibliothek anzuwenden, müssen vorher *iapws* und *iapws97* in das Python-Skript importiert werden.
Ein Beispiel für die Anwendung von *iapws* ist:

```
01. from iapws import iapws97
02. p_sat = iapws97._PSat_T(500)
```

Anhand der Syntax lässt sich der Sättigungsdruck von Wasser bei einer Temperatur von 500 K in MPa berechnen. Die Temperaturgrenze zur Berechnung des Sättigungsdrucks bei der Funktion *iapws97* liegt zwischen 273.15 K ≤ T ≤ 647.096 K.

- **Time**
  Die Bibliothek *time* ermöglicht es, die aktuelle Zeit zu bestimmen und in andere Zeitzonen umzuwandeln. Zur Berechnung der Gasturbine wurde die Bibliothek *time* nur zur Berechnung der Simulationszeit verwendet.

3.4 Python Funktionen

Funktionen werden in Python mit dem Schlüsselwort *def* definiert. Eine Funktion kann ohne Parameter, mit einem Parameter oder mit einer Liste von Parametern definiert werden. Die Parameterliste besteht aus mehreren Werten oder mehreren Bezeichnungen, die durch ein Komma getrennt sind. Die Funktion Definition endet mit einem Doppelpunkt [15]. Es ist entscheidend, den Doppelpunkt anzugeben, sonst zeigt Python einen Fehler. Der Funktionskörper, der auszuführen ist, wenn die Funktion aufgerufen wird, wird in Python durch eine einheitliche Einrückung markiert. Die Funktionsdefinition ist beendet, wenn eine Anweisung erfolgt, die wieder auf derselben Einrückungsstufe steht wie der Kopf der Funktion. Eine zweite Möglichkeit, um die

Funktionsdefinition zu beenden, ist mit der Syntax *return*. Der Rückgabewert in Python lässt sich mit *return* zurückgeben. Es können ein Ergebnis oder mehrere Ergebnisse zurückgegeben werden.

Die erste Funktion *Funktionsname()* (siehe Abbildung 3.4-1) ist eine Standardsyntax, um eine Funktion in Python zu definieren. Die zweite Funktion *Helloworld()* wurde ohne Eingangsparameter definiert. Der Funktionskörper besteht hier nur aus einer Zeile, in der die Print-Funktion benutzt wurde, um ‚Hello World' auszugeben. Bei der dritten Funktion wurden zwei Eingangsparameter a und b definiert. Hier werden die Werte der beiden Parameter summiert und in einer neuen Variablen c gespeichert. Anschließend wird das Ergebnis der Summation mithilfe der Return-Syntax zurückgegeben. Beim Aufrufen der Funktion sind die Werte der Parameter a und b einzugeben.

```
8  def Funktionsname(Parameter):
9      ....
10     ....
11     return
12
13 def Helloworld():
14     print("Hello World")
15     return
16
17 def Summe(a,b):
18     c = a+b
19     return c
```

Abbildung 3.4-1: Python Funktionen

In SPYDER wird eine Funktion aufgerufen, indem die Aufrufsyntax direkt nach der Beendigung der Funktion geschrieben und auf das Run-Symbol ▶ geklickt wird (siehe Abbildungen 3.4-2 und 3.4-3). Zusätzlich müssen alle Eingabeparameter in der definierten Reihenfolge eigegeben werden. Die Ausgabeparameter können entweder direkt zurückgegeben werden oder je nach weiterer Anwendung in einer Variablen gespeichert werden.

```
 8 def Helloworld():
 9     print("Hello World")
10     return
11
12 Helloworld()
```

Abbildung 3.4-2: Aufruf der Funktion Helloworld()

```
12 def Summe(a,b):
13     c = a+b
14     #print(c)
15     return c
16
17 c = Summe(4,5)
```

Abbildung 3.4-3: Aufruf der Funktion Summe()

3.5 Python Variable

Wie in vielen anderen Programmiersprachen müssen Variable in Python nicht deklariert werden. Variable werden hier automatisch deklariert, wenn ihnen einen Wert gegeben werden. In Python unterscheidet man zwischen lokalen und globalen Variablen. Jede Variable, die innerhalb einer Funktion definiert wird, hat automatisch einen lokalen Gültigkeitsbereich. Was mit dieser Variablen innerhalb der Funktion geschieht, hat keinen Einfluss auf andere Variable außerhalb der Funktion, auch wenn diese den gleichen Namen haben. Im Vergleich werden globale Variable außerhalb der Funktion definiert. Diese Variable kann man in jeder Funktion aufrufen.

```
 7 x = 4
 8 def Zahl1():
 9     a = 6
10     b = 7
11     c = a + b
12     print(c)
13     return c
14
15 Zahl1()
```

Abbildung 3.5-1: Globale und lokale Variablen1

```
17 x = 10
18 def Zahl2():
19     a = 20
20     x = 15
21     c = a + x
22     print(c)
23     return c
24
25 Zahl2()
```

Abbildung 3.5-2: Globale und lokale Variablen2

Bei der Funktion Zahl1() ist x eine globale Variable und a, b und c sind lokale Variable. Die Variable x kann in jeder Funktion aufgerufen werden, wobei a, b und c nur innerhalb der Funktion Zahl1() gültig sind.

Bei der zweiten Funktion Zahl2() ist x mit dem Wert 10 eine globale Variable und a, x mit dem Wert 15 und c sind lokale Variable. Die globale Variable x hat keinen Einfluss

auf die Gleichung c = a + x, weil die lokale Variable x innerhalb der Funktion definiert wurde. Falls es keine lokale Variable x gegeben hätte, dann hätte x bei der Gleichung c = a + x einen Wert von 10 gehabt.

3.6 Python Listen

Es gibt zahlreiche Möglichkeiten, wie sich in Python verschiedene Daten zuordnen lassen. Die einfachste Möglichkeit ist die Generierung von Listen. Eine Python-Liste enthält eine aufgeteilte Menge von Daten, die entweder aus gleichen Datentypen oder unterschiedlichen Datentypen besteht [16]. Jedes Element der Liste ist einem Index zugeordnet. Mithilfe dieses Index kann auf die Elemente der Liste je nach Anwendung zugegriffen werden. Um eine Liste in Python zu definieren, werden die Elemente durch Kommas getrennt und von eckigen Klammern eingeschlossen. Einige Beispiele sind:

```
In [1]: A = [1, 2, 3, 4]

In [2]: B = ["eins", "zwei", "drei", "vier"]
```

Die erste Liste A ist eine Liste von vier ganzen Zahlen, wobei die zweite Liste B aus vier Strings besteht. Die folgende Liste C besteht aus zwei ganzen Zahlen, einer Gleitkommazahl und einem String.

```
In [3]: C = [1, "zwei", 3.0, "vier"]
```

Darüber hinaus kann eine Liste auch aus einer Kombination von mehreren Listen, Zahlen und Strings bestehen, wie die folgende Liste.

```
In [4]: D = [[1,2], ["zwei","drei", "vier"], 10]
```

Die Syntax, um auf ein Element aus einer Liste zuzugreifen, ist die Listenbezeichnung mit einer von eckigen Klammern eingeschlossenen Zahl. Die Zahl innerhalb der eckigen Klammern gibt den Index des Elements an, auf das zugegriffen werden soll. Bei Python beginnen die Indices bei 0.

```
In [5]: D[0]
Out[5]: [1, 2]
```

Außerdem besteht die Möglichkeit, eine Liste aus mehreren unterschiedlichen Dateien oder aus mehreren Excel-Tabellen zu definieren. Ein Beispiel für diesen Fall wird im Abschnitt 5.2 im Detail erläutert.

Bei der Berechnung der Gasturbine wurden alle Werte, die während der gesamten Simulation konstant bleiben, als Listen definiert.

3.7 Python Wörterbuch

Das Wörterbuch ist eine weitere Datenstruktur in Python für die Zuordnung von Daten [17]. Anders als in einer Liste ist hier jedes Element einem sogenannten Schlüsselelement zugeordnet. Das Schlüsselelement dient als Index für den späteren Zugriff auf den dem Schlüsselelement zugeordneten Wert. Um ein Wörterbuch zu definieren, werden die Elemente, die durch das Komma getrennt sind, in geschweifte Klammern geschrieben. Jedes Element ist ein Paar von Schlüssel und Wert, das durch Doppelpunkt (:) getrennt wird. Der Wert kann ein String, eine Zahl oder der vom Benutzer definierte Datentyp sein. Unten sind einige Beispiele des Python-Wörterbuchs dargestellt.

```
In [1]: A = {1:"eins", 2:"zwei", 3:"drei"}

In [2]: B = {"eins": "one", "zwei":"two", "drei":"three"}
```

Die Wörterbücher A und B enthalten jeweils drei Elemente mit jeweils einem Schlüssel und seinem Wert. Die Syntax, um auf ein Element aus einem Python-Wörterbuch zuzugreifen, ist die Wörterbuchbezeichnung mit dem von eckigen Klammern eingeschlossenen Schlüssel.

```
In [3]: A[2]
Out[3]: 'zwei'

In [4]: B["drei"]
Out[4]: 'three'
```

Es besteht auch die Möglichkeit, einen Wert, der einem Schlüssel zugeordnet ist, zu aktualisieren und das Wörterbuch mit neuen Elementen zu erweitern.

```
In [5]: A[1] = "EINS"

In [6]: A
Out[6]: {1: 'EINS', 2: 'zwei', 3: 'drei'}


In [7]: B["vier"] = "four"

In [8]: B
Out[8]: {'eins': 'one', 'zwei': 'two', 'drei': 'three',
'vier': 'four'}
```

In dieser Arbeit wird das Wörterbuch zum Einlesen mehrerer Excel-Tabellen, die Stoffwerten von Gasten enthalten, benutzt.

4 Gasturbinenberechnung in Python

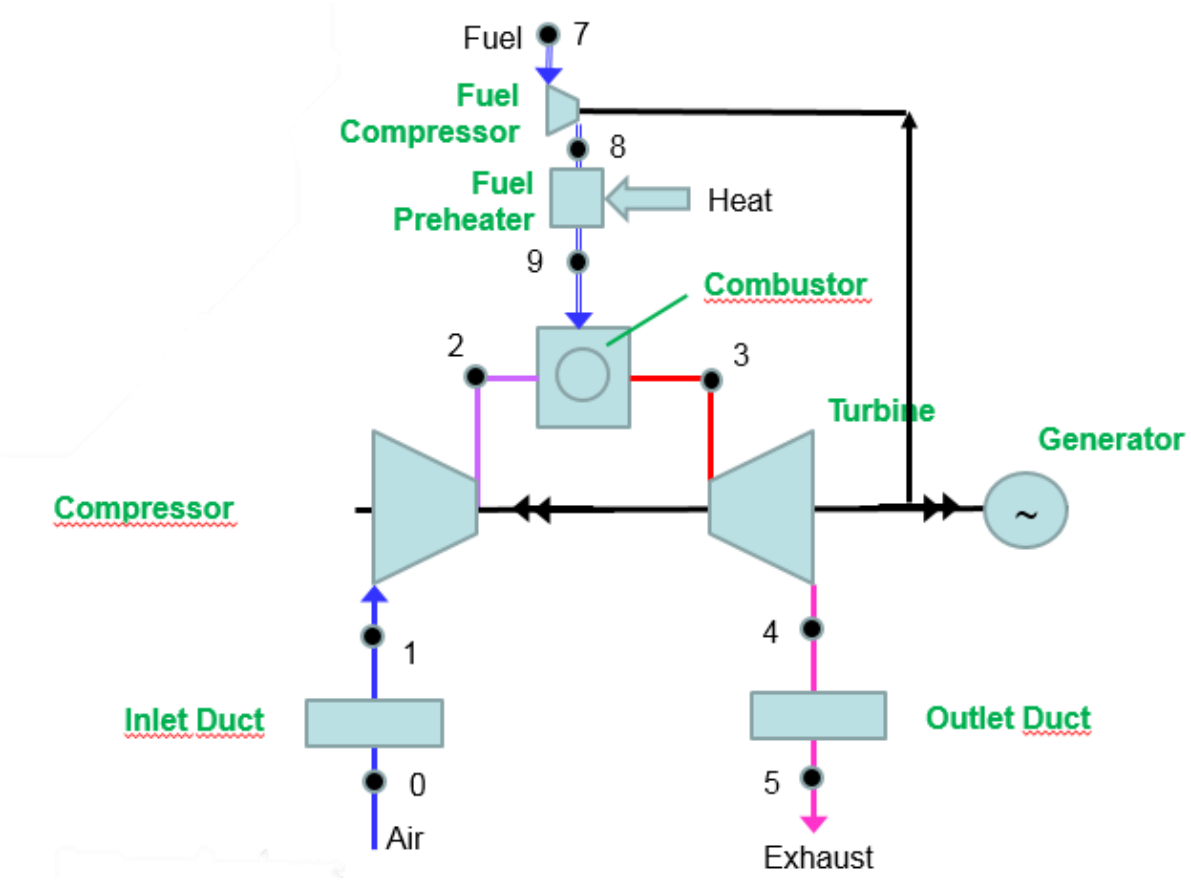

Abbildung 4.1: Gasturbine Anlagenschema

Die Gasturbine wird in drei Schritten simuliert. Jeder dieser Schritte hat eine eigene Python-Datei. Bei der ersten Datei wurden die Daten der Stoffe, die sich im Brennstoff befinden können, aus der Excel-Datei importiert und es wurden vier weitere Hilfefunktionen, die zur Berechnung der thermodynamischen Eigenschaften notwendig sind, geschrieben. Die zweite Datei *Gasturbine.py* enthält die zur Berechnung der Gasturbine notwendigen Algorithmen für jeden Teil der Gasturbine. Diese Datei wurde in mehrere Funktionen aufgeteilt und die tatsächlichen Gasturbinenberechnungen finden hier statt. Bei der dritten Datei *GTkalkulaion.py* wurden alle Funktionen der Datei *Gasturbine.py* aufgerufen und die Ergebnisse dargestellt. Außerdem wurden die Ergebnisse in eine Excel-Datei exportiert.

4.1 hilfefunktionen.py

Als Erstes wurden die zwei Bibliotheken NumPy und Pandas importiert. NumPy wird benutzt, um die Arrays zu definieren. Mithilfe der Bibliothek Pandas werden die Daten aus der Excel-Datei in SPYDER importiert. Als Nächstes wurden die physikalische Konstante und die molare Masse der Stoffe, die sich im Brennstoff befinden können, definiert. Es wird angenommen, dass der Brennstoff nur aus den in den Tabellen 1 und 2 angegebenen Stoffen bestehen kann.

Tabelle 3: Molare Massen 1

Molare Massen					
H2	N2	O2	S	H2O	SO2
2,016 Kg/Kmol	28,016 Kg/Kmol	32,00 Kg/Kmol	32,065 Kg/Kmol	18,016 Kg/Kmol	64,065 Kg/Kmol

Tabelle 4: Molare Massen 2

Molare Massen						
C Gas	CO	CO2	CH4	C2H6	C3H8	C4H10
12,011 Kg/Kmol	28,011 Kg/Kmol	44,011 Kg/Kmol	16,043 Kg/Kmol	30,070 Kg/Kmol	44,097 Kg/Kmol	58,124 Kg/Kmol

Tabelle 5: physikalische Konstanten

Physikalische Konstanten		
g	Avogadros Zahl	Rm
9,81 m/s2	6,023*1023 1/mol	8314,4 KJ/(molK)

Es ist entscheidend, dass die molaren Massen in der angegebenen Reihenfolge von H_2 bis C_4H_{10} definiert sind. Jeder Stoff hat einen festen Index. Der Index der einzelnen Stoffe wurde der Datei Gasturbine.py zugeordnet. Außerdem wurden die Einlassdaten der Stoffe am Lufteintritt in der gleichen Reihenfolge angegeben. Abbildung 4.1-1 zeigt, wie die molaren Massen und die Konstanten in Python definiert wurden. Als Nächstes wurden die vier Python-Listen *O2_fak, H2O_fak, CO2_fak* und *SO2_fak* definiert. Diese sind bedeutsam für die Bestimmung der Rauchgaskonzentration. Die Beschreibung der einzelnen Listen gestaltet sich wie folgt:

O2_fak: Die Liste *O2_fak* beschreibt die Sauerstoffmenge (siehe Tabelle 4), die für die vollständige Verbrennung der 1 Kmol von Gasen aus den Tabellen 1 und 2 benötigt wird. Die Sauerstoffmengen wurden in der in Tabelle 4 angegebenen Reihenfolge von H_2 bis C_4H_{10} definiert.

H2O_fak: Die Liste *H2O_fak* beschreibt die H_2O-Menge (siehe Tabelle 5), die nach der vollständigen Verbrennung der 1 Kmol von Gasen aus den Tabellen 1 und 2 entsteht. Hier ist es ebenfalls entscheidend, die Reihenfolge der Liste *O2_fak* zu berücksichtigen.

CO2_fak: Die Liste *CO_2_fak* beschreibt die CO_2-Menge (siehe Tabelle 5), die nach der vollständigen Verbrennung der 1 Kmol von Gasen aus den Tabellen 1 und 2 entsteht. Die Reihenfolge wird auch hier eingehalten.

SO2_fak: Die Liste *SO_2_fak* beschreibt die SO_2-Menge (siehe Tabelle 5), die nach der vollständigen Verbrennung der 1 Kmol von Gasen aus den Tabellen 1 und 2 entsteht. Die Reihenfolge der Gase wird eingehalten.

Tabelle 4: Sauerstoffmengen zur stöchiometrischen Verbrennung

	Sauerstoffmengen zur vollständigen Verbrennung von 1 Mol von Gasen
H_2	0,5
N_2	0
O_2	0
S	1
H_2O	0
SO_2	0
C Gas	1
CO	0,5
CO_2	0
CH_4	2
C_2H_6	3,5
C_3H_8	5
C_4H_{10}	6,5

Es wird angenommen, dass der Stickstoff (N_2) während der Verbrennung nicht reagiert und kein NO_2 entsteht, d. h., das Rauchgas setzt sich dann aus N_2 (von der Luft), O_2 (Sauerstoffüberschuss, falls $\lambda > 1$), CO_2, SO_2, und H_2O zusammen.

Tabelle 5: Abgasmengen nach der stöchiometrischen Verbrennung

	Durch vollständige Verbrennung von 1 Mol von Gasen entstehende Rauchgasmengen		
	H_2O	CO_2	SO_2
H_2	1	0	0
N_2	0	0	0
O_2	0	0	0
S	0	0	1
H_2O	1	0	0
SO_2	0	0	1
C Gas	0	1	0
CO	0	1	0
CO_2	0	1	0
CH_4	2	1	0
C_2H_6	3	2	0
C_3H_8	4	3	0
C_4H_{10}	5	4	0

Abbildung 4.1-1 zeigt den Code-Abschnitt, in dem die physikalischen Konstanten, die molaren Massen und die vier Listen definiert wurden. Die Variable wurden nicht in einer Funktion, sondern als globale Variable definiert, weil die Variable in mehreren Funktionen dieser Datei verwendet wurden. Die Abbildung zeigt auch das Importieren der Bibliotheken, obwohl sie noch nicht benutzt wurden. Der Grund hierfür ist, dass die Bibliotheken immer am Anfang jeder Datei definiert wurden.

```
import numpy as np
import pandas as pd

gravity, avogadro, Rm = [9.81, 6.023e23, 8314.4]
Mass = [2.016, 28.016, 32.000, 32.065, 18.016, 64.065, 12.011, 28.011,
        44.011, 16.043, 30.070, 44.097, 58.124]
O2_fak = [0.5, 0, 0, 1, 0, 0, 1, 0.5, 0, 2, 3.5, 5, 6.5]
H2O_fak = [1, 0, 0, 0, 1, 0, 0, 0, 0, 2, 3, 4, 5]
CO2_fak = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 3, 4]
SO2_fak = [0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
```

Abbildung 4.1-1: Definition der Listen in Python bei der Berechnung

Als Nächstes wurde die Excel-Datei *ABSMAT.xlsx* mithilfe der Pandas-Funktion *read_excel* eingelesen. Die Datei enthält bei unterschiedlicher Temperatur Stoffwerten der einzelnen Gase aus den Tabellen 1 und 2. Die Excel-Datei besteht aus 13 Reitern. Jeder Reiter besteht aus fünf Spalten, die jeweils die Stoffwerte eines Gases enthalten. Die Reiter sind nach den Namen des Gases bezeichnet. Die erste Spalte ist die Temperatur (K), die zweite Spalte besteht aus den durchschnittlichen spezifischen molaren Wärmekapazitäten (KJ/KmolK) zwischen 0 K und der Temperatur aus der ersten Spalte die dritte Spalte besteht aus den absoluten spezifischen molaren Enthalpien (KJ/Kmol) bei der Temperatur aus der ersten Spalte, die vierte Spalte besteht aus den absoluten sepzifischen isobaren molaren Entropien (KJ/KmolK) bei der Temperatur aus der ersten Spalte und die fünfte Spalte besteht aus den absoluten spezifischen isochoren molaren Entropien (KJ/KmolK) bei der Temperatur aus der ersten Spalte. Es ist entscheidend, dass beim Einlesen der Excel-Datei der Reitername auf keins gestellt ist. Somit wird die gesamte Excel-Datei mit allen 13 Tabellen eingelesen und in Python gespeichert. Dies spart später die Simulationszeit. Andernfalls würde Python jedes Mal beim neuen Material auf Excel zugreifen müssen. Ab hier wurde die Datei *hilfefunktionen.py* in vier Funktionen unterteilt. Die Syntax zum Einlesen der Excel-Datei ist:

```
filepath = 'ABSMAT.xlsx'
sheet = pd.read_excel(filepath, sheet_name=None)
```

4.1.1 absmat_data()

„*def absmat_data(a,b,c,d,material,tmp_ref):*“

In dieser Funktion werden die Temperatur, die absolute spezifische molare Enthalpie, absolute spezifische isobare molare Entropie, absolute spezifische isochore molare Entropie und spezifische molare Wärmekapazität interpoliert und zurückgegeben. Die Eingangsparameter der Funktion absmat_data() sind die Temperatur (*a*), die absolute spezifische molare Enthalpie (*b*), absolute spezifische isobare molare Entropie (*c*), absolute spezifische isochore molare Entropie (*d*), molare Masse von einem Gas aus den Tabellen 1 und 2 (*material*) und die niedrigere Temperatur (*tmp_ref*) für die durchschnittliche spezifische Wärmekapazität. Mindestens ein Parameter aus a, b, c und d darf nicht null sein. Die Materiale und deren molare Massen sind als Python-Wörterbuch definiert. Das Schlüsselwort in diesem Fall sind die molaren Massen der Stoffe aus den Tabellen 1 und 2 und deren Wert sind die dazugehörigen molekularen Bezeichnungen. Somit wurde der Eingangsparameter ‚Material‘ den dazugehörigen Stoffwerten aus der Excel-Tabelle zugeordnet. Je nach Material werden die Stoffwerte vom Python-Dataframe gelesen und, wie bereits erwähnt, in NumPy-Arrays umgewandelt. Die Interpolation der spezifischen Wärmekapazität folgt in zwei Schritten. Als Erstes wird die spezifische Wärmekapazität zwischen 0 K und der gegebenen Temperatur interpoliert und danach zwischen 0 K und der niedrigeren Temperatur (*tmp_ref*) für die durchschnittliche spezifische Wärmekapazität interpoliert. Anschließend wird mithilfe der Gleichung (15) die tatsächliche spezifische molare Wärmekapazität berechnet.

$$c_p\big|_{tmp\_ref}^{T} = \frac{c_p\big|_{0K}^{T} \cdot T - c_p\big|_{0K}^{tmp\_ref} \cdot tmp\_ref}{T - tmp\_ref} \tag{15}$$

4.1.2 matfun_phr()

„*def matfun_phr(tmp_ref, enth, pres, mat):*“

Bei dieser Funktion werden die Temperatur, die absolute spezifische molare Entropie, die molare Masse aus der Gaszusammensetzung und die Gaskonstante bei dem gegebenen Druck, der gegebenen Gaszusammensetzung und der gegebenen Enthalpie berechnet. Hierzu wird die gegebene Gaszusammensetzung und die Temperatur an die Funktion *absmat_data* als Eingangsparameter übergegeben, um die gesuchten Größen zu interpolieren. Der Dämpfungsfaktor wird mit 0,5 definiert und der Abweichungsfehler der Temperatur aus der gegebenen und berechneten Enthalpie wird auf 0,0001 gesetzt. Der Dämpfungsfaktor verlangsamt zwar die Annäherung an die Lösung, verhindert aber, dass man bei der Lösungssuche divergiert.

4.1.3 matfun_ptr()

„*def matfun_ptr(tmp_ref, temp, pres, mat):*“

In dieser Funktion werden die absolute spezifische molare Enthalpie, die absolute spezifische molare Entropie, die molare Masse aus der Gaszusammensetzung, die spezifische molare Wärmekapazität und die Gaskonstante bei dem gegebenen Druck und der gegebenen Temperatur berechnet. Mithilfe der Funktion *absmat_data()* werden die gesuchten Größen interpoliert und zurückgegeben.

4.1.4 matfun_psr()

„*def matfun_psr(tmp_ref, entr, pres, mat):*“

Diese Funktion berechnet bei der gegebenen Entropie, dem Druck und der Gaszusammensetzung die absolute spezifische molare Enthalpie, die Temperatur, die molare Masse aus der Gaszusammensetzung und die Gaskonstante. Es werden zwei Entropien mithilfe der Funktion *absmat_data()* ermittelt, jeweils eine größer und eine kleiner als die gegebene Entropie. Die entsprechenden Temperaturen zu den Entropien werden berechnet. Als Nächstes wird die gesuchte Temperatur zwischen den

beiden Entropien linear interpoliert. Anschließend werden zu der linear interpolierten Temperatur die absolute spezifische molare Enthalpie, die molare Masse aus der Gaszusammensetzung und die Gaskonstante aus der Funktion *absmat_data()* ermittelt.

4.2 Gasturbine.py

Die tatsächliche Berechnung der Gasturbine findet hier statt. Die Datei ist in neun Funktionen unterteilt. Bei jeder Funktion werden die thermodynamischen bzw. molekularen Eigenschaften von jeweils einem Punkt der Gasturbine (siehe Abbildung 4.1) berechnet. Die zu berechnenden Daten werden jeweils einem Index zugeordnet. Die Indizes beschreiben die Position der Ergebnisse in den Ergebnismatrizen. Die Indizes und deren Bezeichnungen sind aus der Tabelle 6 zu entnehmen. Die dort angegebenen Größen sind an jedem Punkt der Gasturbine in Stoffkonzentration sowie in Massenkonzentration zu berechnen. Die Umgebungstemperatur und der Umgebungsdruck werden als 298,15 K bzw. 1,013 Bar angenommen.

Jede Funktion außer der der Verbrennungskammer nimmt folgende Parameter als Eingangsdaten:

IDX:	Der zu berechnende tatsächliche Punkt in der Gasturbine
variables:	Anzahl der zu berechnenden Daten
MatArrayKmol:	Ergebnis Matrix bezogen auf Stoffmengenkonzentration
MatArrayKg:	Ergebnis Matrix bezogen auf Massenkonzentration

Zusätzlich zu den oben genannten vier Variablen werden in der Verbrennungskammer auch die molare Masse der Luft und des Brennstoffs als Eingangsparameter übergegeben.

In jeder der neun Funktionen wird die Exergie bilanziert. Dazu werden die Gleichungen (16) und (17) verwendet. Die Ergebnisse der Bilanzierung werden in eigens dafür angelegten Matrizen gespeichert.

$$e_m = h_m - h_{mUmgebungsluft} - T_{Umgebungsluft} \cdot \left(s_m - s_{mUmgebungsluft}\right) \quad (16)$$

$$e = h - h_{Umgebungsluft} - T_{Umgebungsluft} \cdot \left(s - s_{Umgebungsluft}\right) \quad (157)$$

Tabelle 6: Bezeichnung der einzelnen Indices

Bezeichnung	Index	Beschreibung	Einheit
posF	0	Massenstrom oder Stoffmengen-strom	Kg/s oder Kmol/s
posT	1	Temperatur	K
posP	2	Druck	bar
posV	3	spezifisches Volumen	m^3/Kg oder m^3/Kmol
posH	4	absolute spezifische Enthalpie	KJ/Kg oder KJ/Kmol
posS	5	absolute spezifische Entropie	KJ/KgK oder KJ/KmolK
posE	6	absolute spezifische Exergie	KJ/mol oder Kg/Kmol
posH2	7	Wasserstoffkonzentration	%
posN2	8	Stickstoffkonzentration	%
posO2	9	Sauerstoffkonzentration	%
posSS	10	Schwefelkonzentration	%
posH2O	11	Wasserkonzentration	%
posSO2	12	Schwefeldioxidkonzentration	%
posC	13	Kohlenstoffkonzentration	%
posCO	14	Kohlenstoffmonoxidkonzentration	%
posCO2	15	Kohlenstoffdioxidkonzentration	%
posCH4	16	Methankonzentration	%
posC2H6	17	Ethankonzentration	%
posC3H8	18	Propankonzentration	%
posC4H10	19	Butankonzentration	%

4.2.1 air_inlet()

"*def air_inlet(IDX, variables, MatArrayKmol, MatArrayKg)*:"

Die Funktion *air_inlet* berechnet die Stoffwerte der Umgebungsluft am Lufteintritt (Position 0) der Gasturbine. Der Umgebungszustand der Luft wurde aus einer Excel-Datei

eingelesen. Diese Excel-Datei ist die Datenquelle für die Berechnungen. Dort sind die Luftzusammensetzung und der Luftstrom sowie die Zustandsgrößen Temperatur und Druck gegeben. Die Luftzusammensetzung ist entweder als Massenkonzentration oder als Stoffkonzentration definiert. Die Funktion air_inlet() berechnet die noch fehlende Konzentration für ihren Zwischenschritt. Dazu werden die thermodynamischen Formeln aus der Tabelle 7 verwendet. $\mathrm{M_{Luft}}$ wird mithilfe der angegebenen Luftzusammensetzung berechnet.

Tabelle 7: Umwandlung Massenkonzentration in Stoffkonzentration und umgekehrt

1	$\dot{m}_{Luft} = \dot{n}_{Luft} \cdot M_{Luft}$	Umwandlung vom Stoffmengenstrom zum Massenstrom
2	$\xi_i = \mathrm{x}_i \cdot \frac{M_i}{M_{Luft}}$	Umwandlung von der Stoffkonzentration zur Massenkonzentration
3	$\dot{n}_i = \frac{\dot{m}_{Luft}}{M_{Luft}}$	Umwandlung vom Massenstrom zum Stoffmengenstrom
4	$x_i = \xi_i \cdot \frac{M_{Luft}}{M_i}$	Umwandlung von der Massenkonzentration zur Stoffkonzentration

Als Nächstes müssen die absolute spezifische molare Enthalpie, die absolute spezifische molare Entropie sowie die Gaskonstante mithilfe der Funktion *matfun_ptr()* in lokale Variable zur Weiterverarbeitung geladen werden. Innerhalb der Funktion *matfun_ptr()* werden die absolute spezifische molare Enthalpie und die absolute spezifische molare Entropie berechnet. Die Gaskonstante wird hingegen nur in Massenkonzentration ermittelt. Anschließend wird das spezifische Volumen in Stoff- und Massenkonzentration berechnet. Die Tabelle 8 liefert die Formeln dafür.

Die Zustandsgrößen der Luft werden als globale Variable deklariert, um sie später bei der Exergiebilanzierung benutzen zu können. Die Ergebnisse werden in den entsprechenden Spalten der nullten Zeile in beiden Matrizen gespeichert.

Tabelle 8: Umwandlung der absoluten spezifischen molaren Enthalpie, Entropie und Volumen

1	$h = \frac{h_m}{M_{Luft}}$	Umwandlung der absoluten spezifischen molaren Enthalpie zu der absoluten spezifischen Enthalpie
2	$s = \frac{s_m}{M_{Luft}}$	Umwandlung der absoluten spezifischen molaren Entropie zu der absoluten spezifischen Entropie
3	$v_m = R_m \cdot \frac{T}{p \cdot 100000}$	Bestimmung des spezifischen molaren Volums
4	$v = R \cdot \frac{T}{p \cdot 100000}$	Bestimmung des spezifischen Volums

4.2.2 inlet_duct()

“def inlet_duct(IDX, variables, MatArrayKmol, MatArrayKg):”

Vom Zustand 0 nach Zustand 1 folgt ein Druckverlust von 0,01 Bar. Der Druckverlust ist in der Datenquelle vorgegeben und wird entsprechend eingelesen. Der Druck wird für diesen Zustand angepasst. Der Luftstrom, die Luftzusammensetzung, die Temperatur und die Enthalpie ändern sich nicht und werden deshalb in die zweite Zeile der Ergebnismatrix kopiert. Mithilfe der Funktion *matfun_phr()* wird die absolute spezifische molare Entropie für den neuen Druck aufgerufen und dann in die absolute spezifische Entropie umgewandelt. Die Ergebnisse werden in den entsprechenden Spalten der ersten Zeile in den beiden Matrizen gespeichert. Der Vorgang erfolgt mit leichtem Exergieverlust. Die Exergie wird bilanziert und in den Exergiematrizen gesichert.

4.2.3 air_compressor()

„def air_compressor(IDX, variables, MatArrayKmol, MatArrayKg):“

Die Umgebungsluft tritt jetzt mit einer Temperatur von 298,15 K und einem Druck von 1,003 Bar in den Verdichter ein. Die Luft wird komprimiert und der Druck erhöht sich. Der isentrope Wirkungsgrad sowie das Druckverhältnis sind in der Datenquelle vorgegeben. Die Luftzusammensetzung und der Luftstrom ändern sich nicht. Der Druck am Verdichteraustritt wird mithilfe des Druckverhältnisses berechnet. Bei dem isentropen Vorgang bleibt die absolute spezifische molare Entropie konstant und nach der Gleichung (18) wird die isentrope Temperatur geschätzt.

$$T_{2,isen} = T_1 \cdot \left(\frac{p_2}{p_1}\right)^{\frac{\kappa-1}{\kappa}} \tag{18}$$

Die Funktion *matfun_psr()* liefert die tatsächliche isentrope Temperatur sowie die Enthalpie nach dem isentropen Vorgang. Als Nächstes wird die Enthalpie nach der realen Verdichtung mittels Gleichung (19) berechnet.

$$h_{2,m} = h_{1,m} + \frac{h_{2,m,isen} - h_{1,m}}{\eta_{V,isen}} \tag{19}$$

Da jetzt die tatsächliche Enthalpie nach der Verdichtung bekannt ist, werden mithilfe der Funktion *matfun_phr()* die tatsächliche Temperatur sowie die tatsächliche Entropie nach der realen Verdichtung ermittelt. Die Luft tritt aus dem Verdichter mit einer Temperatur von 742,89 K und einem Druck von 20,06 Bar aus. Die Ergebnisse werden in den entsprechenden Spalten der zweiten Zeile in den beiden Matrizen gespeichert.

4.2.4 fuel_inlet()

„def fuel_inlet(IDX, variables, MatArrayKmol, MatArrayKg):“

In dieser Funktion wird der Zustand des Brennstoffs am Eingang des Brennstoffverdichters aus der Quelldatei eingelesen. Genauso wie bei der Luft sind die Brennstoffzusammensetzung und der Brennstoffstrom sowie die Zustandsgrößen Temperatur und Druck gegeben. Die Brennstoffzusammensetzung ist entweder als Massenkonzentration oder als Stoffkonzentration definiert. Die fehlende Konzentration wird berechnet. Die absolute spezifische molare Enthalpie und die absolute spezifische molare Entropie beim Brennstoffzustand sowie beim Umgebungszustand werden mithilfe der Funktion *matfun_ptr()* ermittelt. Die Entropie und die Enthalpie beim Umgebungszustand werden als globale Variable deklariert, da sie später bei der Exergieberechnung gebraucht werden. Die Brennstoffdaten werden in den entsprechenden Spalten in der siebten Zeile in den beiden Matrizen gespeichert.

Heizwertberechnung

Der spezifische Heizwert Hu eines Brennstoffs gibt an, wie viel Energie in Form von Wärme bei der Verbrennung pro Kilogramm oder pro Mol gewonnen werden kann.[3] Es wird angenommen, dass der Brennstoff und die Luft bei einer Bezugstemperatur zugeführt werden und das Rauchgas nach der Verbrennung wieder auf diese Bezugstemperatur zurückgekühlt werden kann.

Um den Heizwert zu bestimmen, wird die Enthalpie der Luft sowie des Brennstoffs bei einer Umgebungstemperatur von 298,15 K und einem Umgebungsdruck von 1 Bar berechnet. Die Mindestluftmenge sowie die Mindestsauerstoffmenge für die vollständige Verbrennung von 1 Mol Brennstoff werden errechnet. Die Luft- und Brennstoffzusammensetzung wird aus der Datenquelle entnommen. Als Nächstes wird die Rauchgaszusammensetzung nach der stöchiometrischen Verbrennung ($\lambda = 1$) ermittelt. Das Rauchgas setzt sich zusammen aus N_2, SO_2, H_2O und CO_2. Mithilfe der Funktion *matfun_ptr()* und Rauchgaszusammensetzung wird die Enthalpie des Rauchgases bei der Bezugstemperatur sowie dem Bezugsdruck aufgerufen. Der Heizwert des Brennstoffs lässt sich dann nach den Gleichungen 20 und 21 bestimmen.

[3] https://www.energie-lexikon.info/heizwert.html

$$Hu_m = \dot{n}_L \cdot h_{m,L} + \dot{n}_B \cdot h_{m,B} - \dot{n}_{Abgas} \cdot h_{mAbgas} \tag{20}$$

$$Hu = \dot{m}_L \cdot h_L + \dot{m}_B \cdot h_B - \dot{m}_{Abgas} \cdot h_{Abgas} \tag{21}$$

Die gesamte absolute spezifische Exergie am Brennstoffeinlass unter Berücksichtigung des Heizwerts wird mithilfe der Gleichungen 22 und 23 berechnet und in den entsprechenden Spalten der Ergebnismatrizen gespeichert.

$$e_m = h_{m,B} - h_{m,B,Umgebungszustand} - T_{Umgebung} \cdot \left(s_{m,B} - s_{m,B,Umgebungszustand}\right) + Hu_m \tag{22}$$

$$e = h_B - h_{B,Umgebungszustand} - T_{Umgebung} \cdot \left(s_B - s_{B,Umgebungszustand}\right) + Hu \tag{23}$$

4.2.5 fuel_compressor()

„def fuel_compressor(IDX, variables, MatArrayKmol, MatArrayKg, Hum, Hu):“

Mit einer Temperatur von 288,15 K und einem Druck von 6 Bar tritt der Brennstoff in den Brennstoffverdichter ein. Der Verdichter arbeitet mit einem Wirkungsgrad von 85 % und der Druck nach der Verdichtung wird auf 24 Bar vervierfacht. Die Daten des Verdichters werden von der Datenquelle eingelesen. Die isentrope Temperatur nach der Verdichtung wird mithilfe der isentropen Gleichung (18) geschätzt. Die Entropie nach dem isentropen Vorgang bleibt unverändert. Die Funktion *matfun_psr()* liefert dann die tatsächliche Temperatur und die Enthalpie nach der isentropen Verdichtung. Danach wird die tatsächliche Enthalpie nach der realen Verdichtung mithilfe der Gleichung (18) berechnet. Da jetzt die Enthalpie nach der realen Verdichtung vorhanden ist, werden durch die Funktion *matfun_phr()* die tatsächliche Temperatur und die Entropie nach der realen Verdichtung ermittelt. Die Exergie wird unter Berücksichtigung des Heizwerts bilanziert und die Ergebnisse nach der Verdichtung werden in die entsprechenden Spalten der Ergebnismatrizen übergegeben. Nach der Verdichtung hat der Brennstoff eine Temperatur von 416,07 K und wird jetzt in den Vorwärmer geleitet.

4.2.6 fuel_preheater()

„def fuel_preheater(IDX, variables, MatArrayKmol, MatArrayKg, Hum, Hu):“

In dem Brennstoffvorwärmer wird der Brennstoff vorgewärmt und für die Verbrennung vorbereitet. Die Temperatur nach der Vorwärmung von 523,15 K und der minimale Druckverlust von 0,1 Bar sind in der Quelldatei vorgegeben. Da die Temperatur und der Druck nach der Vorwärmung bekannt sind, werden die Enthalpie und die Entropie berechnet. Die Exergie wird unter Berücksichtigung des Heizwerts bilanziert und die Ergebnisse nach der Vorwärmung werden in den entsprechenden Spalten der Ergebnismatrizen gespeichert.

4.2.7 combustion_chamber()

„def combustion_chamber(IDX, variables, MatArrayKmol, MatArrayKg, Mairin, Mfuelin): “

Die Umgebungsluft mit einer Temperatur von 742,89 K und der Brennstoff mit einer Temperatur von 523,15 K treten in die Verbrennungskammer ein. Der Druckverlust von 0,4 Bar wird von der Quelldatei eingelesen. Bei der Verbrennung wird zwischen drei Fällen unterschieden.

Fall 1

Im Fall 1 sind die Rauchgastemperatur nach der Verbrennung und das Verbrennungsluftverhältnis (λ) vorgegeben. Als Erstes wird der Mindestsauerstoffbedarf für die stöchiometrische Verbrennung mithilfe der Luftzusammensetzung und *O2_fak* berechnet und daraus ergibt sich der Mindestluftbedarf nach der Gleichung (24).

$$L_{min} = \frac{1}{x_{O_2,tr.\ Luft}} \cdot O_{min} \tag{24}$$

Da λ und der molare Luftstrom gegeben sind, wird daraus der molare Mindestluftstrom mit der Gleichung (25) ermittelt.

$$\dot{n}_{L,min} = \frac{\dot{n}_L}{\lambda} \tag{25}$$

Nun wird anhand der Gleichung (26) der molare Brennstoffstrom zu dem gegebenen Verbrennungsluftverhältnis (λ) und dem molaren Brennstoffstrom berechnet.

$$\dot{n}_B = \frac{\dot{n}_{L,min}}{L_{min}} \tag{26}$$

Danach wird die Rauchgaszusammensetzung nach der Verbrennung ermittelt. Da jetzt die Rauchgaszusammensetzung bekannt ist, werden mithilfe der Funktion *matfun_ptr(),* die Enthalpie und die Entropie des Rauchgases aufgerufen. Als Eingangsparameter werden neben der Rauchgaszusammensetzung, der Druck und die Temperatur nach der Verbrennung sowie die Vergleichstemperatur zur Bestimmung der spezifischen Wärmekapazität übergegeben.

Fall 2

Beim zweiten Fall sind der thermische Verlust und das Verbrennungsluftverhältnis (λ) gegeben. Der thermische Verlust ist als Prozentsatz angegeben. Mithilfe des thermischen Verlustes lässt sich die absolute spezifische molare Enthalpie nach der Verbrennung mittels Gleichung (27) berechnen.

$$h_{3,m} = \frac{(\dot{n}_2 \cdot h_{2,m} + \dot{n}_9 \cdot h_{9,m}) \cdot \left(1 - \frac{\dot{Q}_v}{100}\right)}{\dot{n}_3} \tag{27}$$

Das Verbrennungsluftverhältnis (λ) ist in diesem Fall dasselbe wie beim ersten Fall. Deshalb werden die Rauchgaszusammensetzung und der molare Rauchgasstrom aus Fall 1 übernommen. Es wird eine Temperatur nach der Verbrennung geschätzt und zusammen mit der absoluten spezifischen molaren Enthalpie, dem Druck nach der Verbrennung und der Rauchgaszusammensetzung an die Funktion *matfun_phr* übergegeben, um die absolute spezifische molare Entropie und die tatsächliche Temperatur nach der Verbrennung zu bestimmen.

Fall 3

Im dritten Fall sind die Rauchgastemperatur nach der Verbrennung und der thermische Verlust gegeben. Zu der gegebenen Rauchgastemperatur von 1650 K muss das Verbrennungsluftverhältnis (λ) geschätzt werden. Dieser Fall kommt dem Einsatz einer

Gasturbine in der Wirklichkeit nahe. Als Erstes wird die Temperatur nach der Verbrennung geschätzt. Danach wird für das Verbrennungsluftverhältnis (λ) = 1 und für das Verbrennungsluftverhältnis (λ) = 6 die Rauchgaszusammensetzung berechnet. Die Enthalpie nach der Verbrennung ergibt sich dann mit der Gleichung (26), da der thermische Verlust bekannt ist. Als Nächstes wird mithilfe der Funktion *matfun_phr()* die tatsächliche Temperatur nach der Verbrennung jeweils für das Verbrennungsluftverhältnis (λ) = 1 und für das Verbrennungsluftverhältnis (λ) = 6 ermittelt. Danach wird die Steigung der Rauchgastemperatur bezogen auf das Verbrennungsluftverhältnis (λ) mit der Gleichung (28) berechnet und das Verbrennungsluftverhältnis (λ) für die gegebene Rauchgastemperatur mit der Gleichung (29) geschätzt. λ_6 und λ_1 sind jeweils 6 und 1 und T ist die gegebene Rauchgastemperatur nach der Verbrennung.

$$T_{steigung} = \left| \frac{T_{\lambda=6} - T_{\lambda=1}}{\lambda_6 - \lambda_1} \right| \tag{168}$$

$$\lambda_{Schätzwert} = 1 + (\lambda_6 - \lambda_1) \cdot \frac{T - T_{\lambda=6}}{T_{\lambda=1} - T_{\lambda=6}} \tag{29}$$

Verbrennungsluftverhältnis (λ) berechnung

Da es jetzt einen Schätzwert des Verbrennungsluftverhältnisses (λ) gibt, wird für dieses λ die Rauchgaszusammensetzung berechnet und daraus die Temperatur nach Verbrennung ($T_{schätzwert}$) ermittelt. Falls die für das geschätzte λ ermittelte Temperatur nach der Verbrennung mit der gegebenen Temperatur nicht übereinstimmt, wird für die ermittelte Temperatur ($T_{schätzwert}$) wieder ein neues Verbrennungsluftverhältnis (λ) mit der Gleichung (30) geschätzt. Der Abweichungsfehler zwischen der für das geschätzte λ ermittelten Temperatur ($T_{schätzwert}$) und der gegebenen Temperatur (T) wird maximal auf 0,001 definiert.

$$\lambda_{Schätzwert} = \lambda_{Schätzwert} + \cdot \frac{T_{schätzwert} - T}{T_{steigung}} \tag{30}$$

Nun wird wieder die Rauchgaszusammensetzung für das neue Verbrennungsluftverhältnis (λ) berechnet und anschließend die Temperatur ($T_{schätzwert}$) nach der Verbrennung ermittelt. Es wird wieder geprüft, ob der Abweichungsfehler zwischen der neuen ermittelten Temperatur ($T_{schätzwert}$) und der gegebenen Temperatur (T) höher als 0,001 ist. Falls der Abweichungsfehler höher als 0,001 ist, wird noch einmal ein neues Verbrennungsluftverhältnis (λ) mit der Gleichung (30) berechnet und erneut iteriert. Die Iteration wird so lange fortgeführt, bis der Abweichungsfehler weniger als 0,001 beträgt oder die Iterationsanzahl 100 erreicht. Bei jeder Iteration werden das neue Verbrennungsluftverhältnis (λ) und die neue Temperatur nach der Verbrennung mithilfe des bei der letzten Iteration berechneten Schätzwerts bzw. der bei der letzten Iteration ermittelten Temperatur berechnet (siehe Gleichung (30)). Wenn schließlich der Abweichungsfehler geringer als 0,001 ist, wird die Iteration nicht weitergeführt und das neue Verbrennungsluftverhältnis (λ) dieser Iteration wird als das tatsächlich gesuchte Verbrennungsluftverhältnis (λ) angenommen. Abbildung 4.7-1 zeigt den Zusammenhang zwischen der Brennkammeraustrittstemperatur und dem Verbrennungsluftverhältnis (λ). Bei der gegebenen Brennkammeraustrittstemperatur von 1650 K ergibt sich ein Verbrennungsluftverhältnis (λ) von ca. 2,4. Der Brennstoffstrom und die Rauchgaszusammensetzung für dieses λ werden berechnet. Die absolute spezifische molare Entropie wird mithilfe der Funktion *matfun_ptr()* ermittelt. Die Ergebnisse werden in den entsprechenden Spalten und Zeilen der Ergebnismatrizen gespeichert. Außerdem werden die Enthalpie und die Entropie des Rauchgases beim Umgebungszustand ermittelt und als globale Variable deklariert, da diese Größen später bei der Exergiebilanzierung verwendet werden.

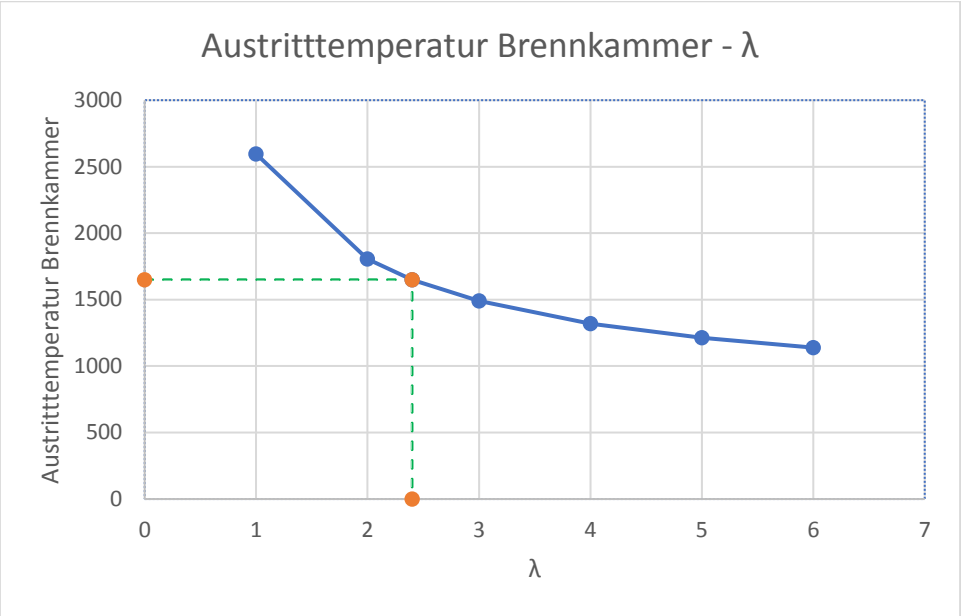

Abbildung 4.7-1: Austritttemperatur Brennkammer und Verbrennungsluftverhältnis

4.2.8 turbine()

„def turbine(IDX, variables, MatArrayKmol, MatArrayKg):“

Das heiße Abgas tritt jetzt in die Turbine ein. Hier wird es entspannt und die Energie gewonnen. Der isentrope Wirkungsgrad und der Druckverlust des Rauchgasauslasses werden von der Quelldatei eingelesen. Da das Rauchgas am Austritt des Rauchgasauslasses Umgebungszustand hat, wird der Druck vor dem Rauchgasauslass (Punkt 4) berechnet. Dies ist zugleich der Druck am Ausgang der Turbine. Die absolute spezifische molare Entropie bleibt bei der isentropen Entspannung konstant. Mit der Gleichung (17) wird die Temperatur nach der isentropen Entspannung geschätzt. Die tatsächliche Temperatur und die absolute spezifische molare Enthalpie nach der isentropen Entspannung werden mit der Funktion *matfun_psr()* ermittelt. Als Nächstes wird die absolute spezifische molare Enthalpie nach der realen Entspannung mittels Gleichung (31) berechnet.

$$h_{4,m} = h_{3,m} - (h_{3,m} - h_{4,m,isen}) \cdot \eta_{,isen} \tag{31}$$

Mit der berechneten absoluten spezifischen molaren Enthalpie und dem Druck am Turbinenaustritt werden die Temperatur und die absolute spezifische molare Entropie

nach der realen Entspannung ermittelt. Die Exergie am Turbinenaustritt wird bilanziert und die Ergebnisse werden in den entsprechenden Spalten der vierten Zeile der beiden Matrizen gespeichert.

4.2.9 outlet_duct()

„def outlet_duct(IDX, variables, MatArrayKmol, MatArrayKg):“

Das heiße Rauchgas wird nach der Turbine über einen Auslass in die Umgebung geleitet. Das erfolgt mit einem leichten Druckverlust von 0,02 Bar. Die Temperatur und die absolute spezifische Enthalpie bleiben unverändert. Die absolute spezifische molare Entropie wird mithilfe der Funktion *matfun_phr()* berechnet und in die absolute spezifische Entropie umgewandelt. Die Ergebnisse werden in den entsprechenden Spalten der beiden Matrizen gespeichert.

4.3 GTKalkulation.py

In dieser Datei werden die Funktionen aus der Datei *Gasturbine.py* in der Reihenfolge von der ersten Funktion zu der letzten aufgerufen. Die Ergebnisse der einzelnen Funktionen werden der darauffolgenden Funktion als Parameter übergegeben. Der Brennstoffstrom wird zu dem gegebenen oder berechneten Verbrennungsluftverhältnis an die Stellen 7, 8 und 9 in der Abbildung 4.1 angepasst. Die Energie und die Exergie werden für das gesamte System bilanziert und die Verluste werden berechnet. Anschließend werden die Ergebnisse in eine Excel-Datei exportiert. Die Ergebnismatrizen werden vor dem Exportieren von NumPy-Arrays zu Pandas-Dataframes umgewandelt. Pandas-Dataframes bieten eine bessere Möglichkeit zur Formatierung des Arrays vor dem Exportieren in die Excel-Datei.

5 Energiebilanzierung

Der Energiestrom in der Gasturbine ist in der Abbildung 5.1 dargestellt. Die Gesamtenergie des Systems ergibt sich aus der Energie vom Brennstoff und der Wärmezufuhr am Brennstoffvorwärmer. Der Wirkungsgrad der Gasturbine ergibt 39,18 %. Ein Teil der Turbinenleistung wird zum Antreiben der beiden Verdichter (Luft- und Brennstoffverdichter) benötigt. Dieses entspricht 37,02 % der gesamten Energie.

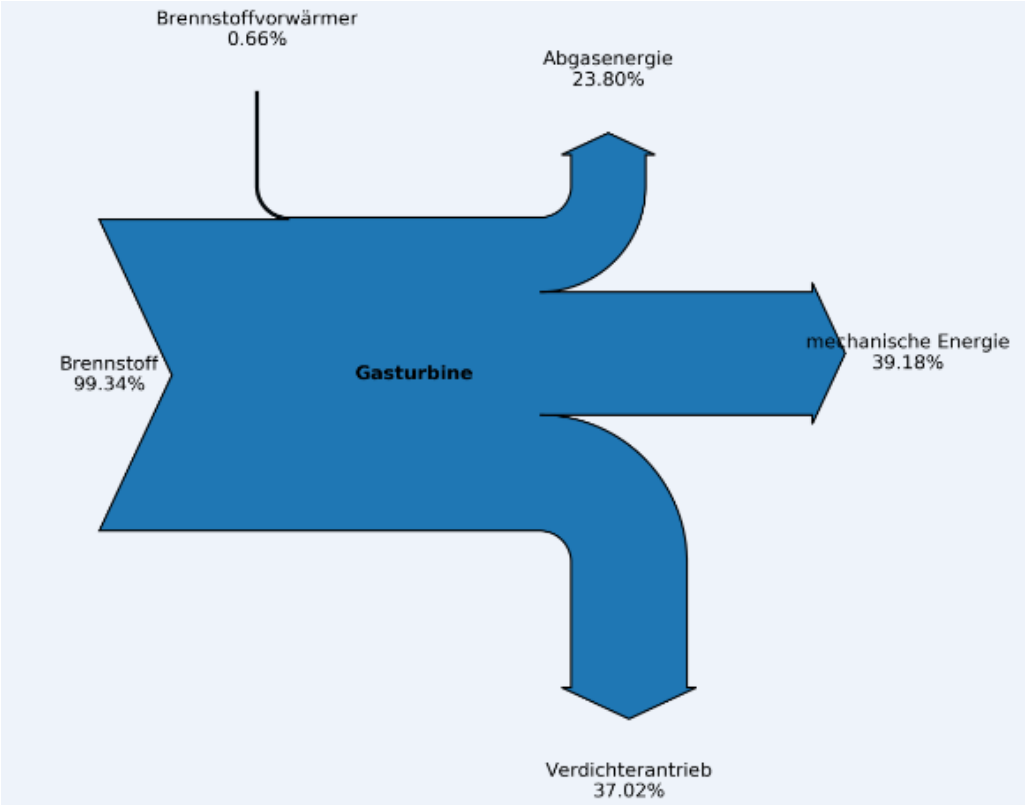

Abbildung 5.1: Energiestrom in der Gasturbine

6 Vergleich mit MATLAB

6.1 Unterschied

- **Importieren von Bibliotheken**
  Die Bibliotheken sind ein sehr wichtiger Teil der Programmierung in Python. In MATLAB sind die mathematischen Funktionen vordefiniert, wobei in Python für jede Anwendung Bibliotheken benötigt werden. Es ist zu empfehlen, dass alle notwendigen Bibliotheken global am Anfang importiert werden, damit sie später in jeder Funktion verwendet werden können.

- **Indices**
  In Python beginnen die Indices von den sequentiellen Datentypen bei 0 wobei in MATLAB bei 1. Die Elemente eines sequentiellen Datentyps haben eine definierte Reihenfolge und werden in Python mit eckigen Klammern indiziert.

- **Excel-Datei einlesen**
  In MATLAB wird jede Tabelle der Excel-Datei jeweils als MATLAB-Datei gespeichert und jede einzelne Datei später in der Funktion aufgerufen. In Python wird das mehrmalige Aufrufen der externen Dateien vermieden, weil die Tabellen der Excel-Datei mit Hilfe der Pandas Bibliothek eingelesen werden und in effizienter Weise als Arrays gespeichert. Somit sind alle notwendigen Daten innerhalb der Funktion vorhanden und die Simulationszeit ist kürzer im Vergleich zu MATLAB.

- **Programmstruktur**
  In Python ist der Programmcode nur in 3 Python-Dateien aufgeteilt. Jede Datei hat eine bestimmte Aufgabe während der Simulation. Der Programmcode von jeder Datei bildet sich dann wieder mit mehreren Funktionen zusammen. Es ist zu empfehlen, dass die Variablen, die in jeder Funktion der gleichen Datei verwendet werden, am Anfang der Datei als globale Variablen zu definieren. Im Vergleich besteht der MATLAB Programmcode aus 30 Dateien. Jede Funktion,

sowie die Tabellen aus der Excel-Datei, sind jeweils als MATLAB-Datei gespeichert.

6.2 Ergebnisse

Die Ergebnisse von den beiden Berechnungsmethoden sind nahezu gleich. In der Tabelle 4 sind die absolute spezifische Enthalpie, die absolute spezifische Entropie und die absolute spezifische Exergie zwischen den beiden Methoden verglichen.

Tabelle 9: Vergleich der Ergebnisse beim Fall 3 in der Brennkammer

	MATLAB			**Python**		
	Spez. Abs. Enthalpie kJ/kg	Spez. Abs. Entropie kJ/kgK	Spez. Abs. Exergie kJ/kg	Spez. Abs. Enthalpie kJ/kg	Spez. Abs. Entropie kJ/kgK	Spez. Abs. Exergie kJ/kg
Air Inlet	-100,179	6,712	0,000	-100,179	6,712	0,000
Compressor Inlet	-100,179	6,715	-0,827	-100,179	6,715	-0,827
Compressor Outlet	349,273	6,807	422,124	349,201	6,807	422,079
Combustor Outlet	242,894	7,992	1309,770	242,826	7,992	1309,690
Turbine Outlet	-673,966	8,134	351,978	-673,993	8,134	351,940
Exhaust Outlet	-673,966	8,140	350,318	-673,993	8,140	350,280
Fuel Inlet	-4692,866	10,574	50308,966	-4692,866	10,574	50308,966
Fuel Compressor Outlet	-4406,638	10,695	50560,412	-4406,640	10,695	49181,994
Fuel Preheater Outlet	-4074,205	11,408	50687,261	-4074,205	11,408	50687,261

6.3 Simulationszeit

Die Simulationszeit wurde (siehe Tabelle 10) aus den 5 Simulationen jeweils mit MATLAB und Python berechnet. Die durchschnittliche Simulationszeit mit der MATLAB beträgt 31,5 Sekunden und im Vergleich mit Python nur 1,16 Sekunden. Ein Grund dafür ist, dass in der MATLAB-Simulation, die Stoffwerten von einzelnen Stoffen als

MATLAB-Dateien gespeichert sind. Bei der Simulation werden diese Dateien 442 Mal aufgerufen und somit wird die Simulationszeit verlangsamt. In Python sind die Tabellen innerhalb der Funktion als Arrays gespeichert. Außerdem beträgt die durchschnittliche Zeit zum Exportieren der Ergebnisse in einer Excel-Datei in MATLAB 22,74 Sekunden.

Tabelle 10: Vergleich der Simulationszeit

Simulationszeit in Sekunden						
	1	2	3	4	5	Durchschnitt
MATLAB	31,14	31,35	32,82	31,28	30,92	31,5
Python	1,32	1,13	1,11	1,1	1,12	1,16

7 Fazit

Ziel dieser Bachelorarbeit ist es, mit Python eine Alternative zur thermodynamischen Berechnung von Gasturbinen in MATLAB zu Programmieren und anschließend zu Evaluieren. Diese Zielsetzung wurde erreicht. Vorhanden war eine Simulation in MATLAB, welche als Grundlage für die Simulation in Python diente. Diese Arbeit hat aufgezeigt, dass die Ausführung des Python-Programms um ein Vielfaches schneller ist als das vorhandene MATLAB-Programm. Zudem ist die Verwendung von Python kostenlos, während für MATLAB eine Lizenz erworben werden muss. Aufgrund dieser beiden Argumente lässt sich eine klare Empfehlung für Simulationen in Python schlussfolgern.

VI. Anhang

Beiliegend ist eine CD mit folgenden Inhalten:

Ordner	**Inhalt**
1 Dokumentation	Enthält dieses Dokument als Word- und PDF-Datei
2 Python	Dieser Ordner enthält den Python Programmcode inklusive der Excel-Dateien mit Stoffwerten von Gasen und Eingangswerten.
3 MATLAB	Dieser Ordner enthält den MATLAB Programmcode inklusive der Excel-Dateien mit Stoffwerten von Gasen und Eingangswerten.

VII. Literaturverzeichnis

[1] http://web.mit.edu/aeroastro/labs/gtl/early_GT_history.html

[2] Christof Lechner, Jörg Seume Hrsg.: Stationäre Gasturbinen, S. 4.

[3] Walter Bitterlich, Ulrich Lohmann: Gasturbinenanlage, S. 2.

[4] https://ipfs.io/ipfs/QmXoypizjW3Wkn-FiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Thermodynamic_cycle.html

[5] Prof. Dr.-Ing. Franz Vinnemeier: Vorlesungsskript Gasturbinen, HAW Hamburg

[6] https://www.asue.de/sites/default/files/asue/themen/gasturbinen/1994/broschueren/11_09_94_1994_grundlagen_technik_betrieb.pdf

[7] https://www.python.org/about/apps/

[8] https://www.anaconda.com/distribution/

[9] https://en.wikipedia.org/wiki/Spyder_(software)

[10] https://docs.anaconda.com/anaconda/user-guide/tasks/integration/spyder/

[11] https://www.python-kurs.eu/numpy.php

[12] https://en.wikipedia.org/wiki/Pandas_(software)

[13] https://de.wikipedia.org/wiki/International_Association_for_the_Properties_of_Water_and_Steam

[14] http://www.iapws.org/

[15] https://pythonbuch.com/funktion.html

[16] https://docs.python.org/3.1/tutorial/datastructures.html

[17] https://docs.python.org/3.1/tutorial/datastructures.html#dictionaries

[18] https://publik.tuwien.ac.at/files/PubDat_196452.pdf