DLR-IB-FT-BS-2022-194

Entwicklung eines Skripts zur Ausgabe von Wettervorhersagedaten

Interner Bericht
Hochschulschrift

Autor: Martin Jansen

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Institutsbericht
DLR-IB-FT-BS-2022/194

Entwicklung eines Skripts zur Ausgabe von Wettervorhersagedaten

Martin Jansen

Institut für Flugsystemtechnik
Braunschweig

51 Seiten
9 Abbildungen
1 Tabellen
77 Referenzen

Deutsches Zentrum für Luft- und Raumfahrt e.V.
Institut für Flugsystemtechnik
Abteilung Flugdynamik und Simulation

Stufe der Zugänglichkeit: I, Allgemein zugänglich: Der Interne Bericht wird elektronisch ohne Einschränkungen in ELIB abgelegt. Falls vorhanden, ist je ein gedrucktes Exemplar an die zuständige Standortbibliothek und an das zentrale Archiv abzugeben.

Braunschweig, den 31.01.2022

Unterschriften:

Institutsdirektor:	Prof. Dr.-Ing. S. Levedag	_____
Abteilungsleiter:	Dr.-Ing. H. Duda	_____
Betreuer:	Dr.-Ing. F. Abdelmoula	_____
Verfasser:	Martin Jansen	_____

ZÜRCHER HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN

Zentrum für Aviatik ZAV

MASTER OF SCIENCE IN ENGINEERING

VERTIEFUNGSARBEIT 1

Entwicklung eines Skripts zur Ausgabe von Wettervorhersagedaten

Autor:
Martin Jansen

Betreuer:
Dr. Julien Anet, ZAV
Dr. Fethi Abdelmoula, DLR

Industriepartner:
Martin Gerber, SWISS
Jan Vetsch, SWISS

31. Januar 2022

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

DECLARATION OF ORIGINALITY

Specialization project at the School of Engineering

DECLARATION OF ORIGINALITY
Specialization project at the School of Engineering

By submitting this specialization project, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Name Student:**
Winterthur, 31.01.2022	Martin Jansen

ZÜRCHER HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN

Zusammenfassung

School of Engineering
Zentrum für Aviatik ZAV

Vertiefungsarbeit 1

Entwicklung eines Skripts zur Ausgabe von Wettervorhersagedaten

von Martin Jansen

Die Luftfahrtbranche wird sich in den nächsten Jahren mit verschiedenen Problemen wie Lärm- und CO_2-Ausstoss beschäftigen müssen. Um diese anzugehen, sind bereits heute viele Projekte initiiert worden. Eines davon ist das Low Noise Augmentation System LNAS des DLR. Es soll Linienpilot:innen dabei helfen, Anflüge von Verkehrsflugzeugen zu optimieren und dadurch Lärm und Kerosinverbrauch zu reduzieren. Die Funktionsweise des LNAS-Systems ist stark von den Wetterbedingungen abhängig. Da die Kurzstreckenflotte von SWISS, wo das System getestet werden soll, nicht über Internet an Bord verfügt, werden nur die vom Flugzeug gemessenen Wetterparameter benutzt. Diese bilden aber die Atmosphäre nicht vollständig ab.

Ziel dieser Arbeit war deshalb, ein Tool zu entwickeln, mit dem Wettervorhersagedaten vor dem Start der Flugzeuge dem LNAS zur Verfügung gestellt werden. Es wurde ein Pythonskript erstellt, welches Daten der ICON-EU- und ICON-DE-Modelle des DWD herunterlädt und ausliest. Als Eingabe werden die Koordinaten, die Höhe und optional die Zeit der interessierten Punkte benötigt. Die Ausgabe erfolgt mit `print`-Statements sowie in einer csv-Datei. Durch Vergleichsmessungen realer Flüge konnte gezeigt werden, dass die ausgelesenen Daten realistisch sind und nur geringfügig von der tatsächlichen Atmosphäre abweichen. In nachfolgenden Arbeiten soll die Validierung ausgeweitet werden. Dabei soll vor allem geprüft werden, ob sich die Performance des LNAS-Systems verbessert, wenn die Vorhersagedaten benutzt werden.

ZURICH UNIVERSITY OF APPLIED SCIENCES

Abstract

School of Engineering
Centre for Aviation ZAV

Specialisation project 1

Entwicklung eines Skripts zur Ausgabe von Wettervorhersagedaten

by Martin Jansen

Noise and CO_2 pollution will be among the greatest challenges the aviation sector will have to face in the near future. Various projects have already been started to tackle these tasks, one of them being the Low Noise Augmentation System LNAS by DLR. It aims to help reduce noise and fuel burn of airline traffic by telling pilots how to optimally fly their approach. The system is therefore greatly dependent on the current weather situation. However, for the case of short haul aircraft of SWISS, upload of weather data inflight is not possible due to lack of connectivity on board. Measured data provided to the LNAS will only result in an incomplete description of the atmosphere.

The scope of this project was therefore to develop a tool which can provide weather forecast data to the LNAS system prior to the aeroplanes departure. For this, a Python script downloading and reading data from the ICON-EU and ICON-D2 models by DWD has been programmed. It requires the coordinates of points of interest, as well as their altitude and optionally a timeframe as input. The output is displayed via `print`-statements, plus it is saved in a csv-file. To show the accuracy of the acquired data, comparisons to real flight measurements have been made. Further validation including implementation into the LNAS system shall be part of future projects.

Danksagung

An dieser Stelle möchte sich der Autor ganz herzlich bei seinem Betreuer Dr. Julien Anet vom ZAV bedanken. Seine wertvollen Inputs als Mentor, seine guten Ratschläge bei Programmierfragen sowie seine Flexibilität in der Terminfindung auch am Wochenende haben dem Autor enorm bei der Entwicklung dieser Arbeit geholfen. Ausserdem bedankt sich der Autor beim Zweitbetreuer Dr. Fethi Abdelmoula vom DLR für seine hilfreichen Gedanken.

Weiter gebührt grosser Dank Martin Gerber und Jan Vetsch von der SWISS, sowie der gesamten Firma, für die Unterstützung und die Daten zur Validierung dieser Arbeit. Zuletzt möchte sich der Autor auch bei seinem ehemaligen Mitstudenten Stefan Fluck vom Zentrum für Aviatik bedanken. Er stand zu jeder Tages- und Nachtzeit für jegliche Fragen zur Verfügung.

Inhaltsverzeichnis

Zusammenfassung ii

Abstract iii

Danksagung iv

1 Einleitung 1

2 Theoretische Grundlagen 2
2.1 Numerische Wettervorhersagen . . . 2
2.1.1 Globalmodelle . . . 3
Datenassimilation . . . 3
Modellberechnung . . . 5
Postprocessing . . . 6
2.1.2 Ensemblemodelle . . . 6
2.1.3 Lokalmodelle . . . 7
2.1.4 ICON und COSMO-Modelle . . . 7
2.2 Softwarepakete und Dateiformate . . . 8
2.2.1 Grib2 . . . 8
2.2.2 ecCodes . . . 9
2.3 LNAS . . . 9

3 Methodik 11
3.1 Umgebung und Programme . . . 11
3.1.1 Windows oder Linux . . . 11
3.1.2 Matlab oder Python . . . 11
3.1.3 Weitere benötigte Software . . . 11
3.2 Entwicklung . . . 12
3.3 Performance . . . 14
3.4 Validierung . . . 15

4 Resultate 17
4.1 Python-Skript . . . 17
Initialisierung . . . 17
for-Schleife über die Punkte . . . 18
for-Schleife über die Variablen . . . 18
4.2 Output . . . 19
4.3 Performance . . . 20
4.4 Validierung . . . 21
4.5 Spinoff ICON-Downloader . . . 23

5 Diskussion **25**
Skript 25
Logikfehler im Ablauf 26
Performance 27
Validierung 27

A Flowchart **29**

B Source Code **32**

Abbildungsverzeichnis **39**

Literatur **40**

Kapitel 1

Einleitung

„For once you have tasted flight, you will forever walk the earth with your eyes turned skyward; for there you have been, and there you long to return.“

Auch wenn diese viel zitierte Aussage wohl nicht wie oftmals behauptet von Leonardo Da Vinci stammt, sondern aus einem Film über diesen, zeigt sie trotzdem deutlich, welche Begeisterung die Aviatik auslösen kann. Man sagt, wer einmal vom Flugfieber gepackt wurde, kommt nicht mehr davon los. Dies trifft auch auf den Autor dieser Arbeit zu. Aus diesem Grund erfüllt es ihn mit Freude und auch Stolz, dass er einen Beitrag an ein Projekt leisten kann, das der Branche zugutekommt und einen Versuch darstellt, sich den komplexen Problemen der aktuellen Zeit zu anzunehmen.

Klimawandel und soziale Verantwortung sind zwei Begriffe, die viele Betriebe vor grosse, aber nicht unlösbare Herausforderungen stellen. Gleichzeitig sind durch die Covid-19-Pandemie Mittel und Ressourcen beschränkt. Das LNAS-Projekt des DLR ist daher ein Weg, um diesen Problemen anhand von Technik entgegenzusteuern. Es hat das Ziel, den Anflug von Verkehrsflugzeugen zu optimieren, indem sie weniger Lärm verursachen, weniger Treibstoff verbrauchen und damit auch weniger Kosten verursachen.

Als Linienpilot hat der Autor wie viele seiner Berufskolleg:innen ausserdem den Ehrgeiz entwickelt, die Operation eines Flugzeuges und seine Arbeitsweise so effizient wie möglich zu gestalten. Wenn weniger Treibstoff verbraucht wird als geplant, und der Anflug gelingt, ohne dass unnötig Gas gegeben werden muss, dann beflügelt dies zusätzlich. Diese intrinsische Motivation hat dazu beigetragen, dass dem Autor dieser Beitrag zum LNAS-Projekt viel Freude bereitet hat.

Kapitel 2

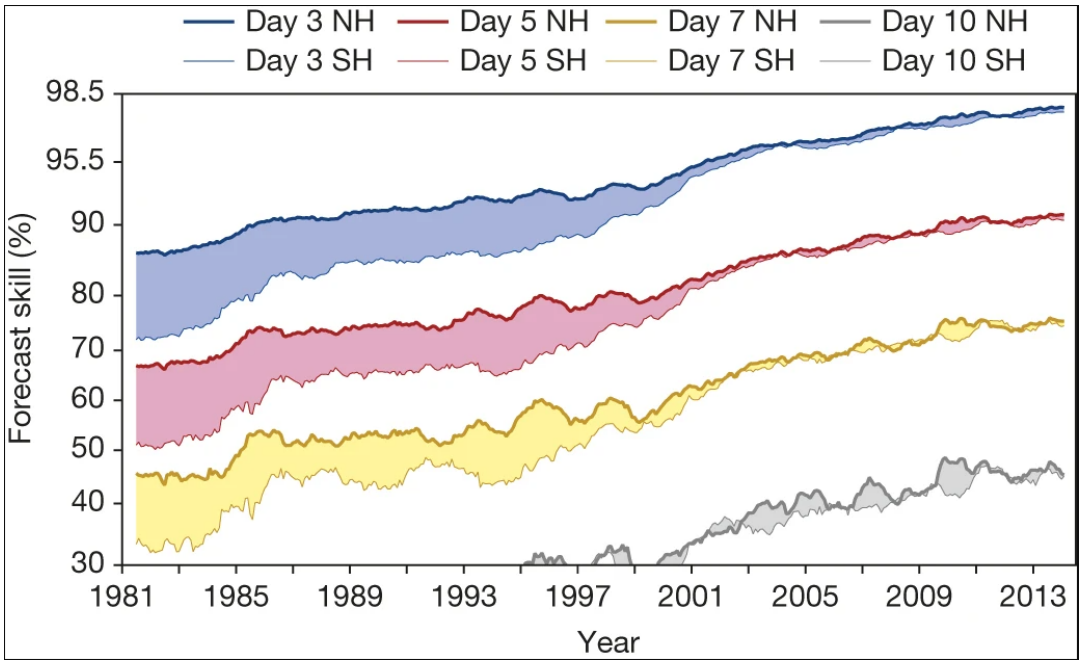
Theoretische Grundlagen

2.1 Numerische Wettervorhersagen

Das Wetter spielt in unser aller Leben eine zentrale Rolle. Die Landwirtschaft (Cogato et al., 2019) oder die weltweite Energieversorgung und Stromproduktion (Staffell & Pfenninger, 2018) hängen davon ab, es beeinflusst unseren Lebensstil und unsere Kleiderwahl oder dient ganz einfach als Thema für Smalltalk. Auch die Luftfahrt ist sehr stark abhängig vom Wetter (Barleben et al., 2020; Boyd, 2017; Gultepe et al., 2019). Insbesondere auf die Vorhersage des Wetters wird daher seit jeher ein grosser Fokus gelegt. Unzählige staatliche und private Wetterdienste rund um den Globus stellen immer akkuratere und weitsichtigere Wetterprognosen her (Bauer et al., 2015). Neben den „gewöhnlichen" Wettervorhersagen für den Privatverbraucher werden dabei vorallem Produkte für spezifische Dienste wie See- und Luftfahrt sowie auf Katastrophenschutzeinrichtungen entwickelt, beispielsweise Frühwarnsysteme für Unwetter oder Starkregen (DWD, 2021a). Um solche Prognosen zu erstellen bedarf es mathematischer Hilfsmittel, namentlich numerischer Wettermodelle (engl.: Numerical weather prediction, NWP).

Die Anfänge numerischer Wettermodelle gehen zurück bis ins frühe zwanzigste Jahrhundert. Richardson (1922) entwarf das erste Basismodell. Da damals aber die nötige Rechenleistung noch nicht vorhanden war, dauerte es bis in die 1950er Jahre, bis sich mit dem Entstehen der ersten Supercomputer NWP im operationellen Betrieb erstmals etablieren konnten (Lynch, 2006). Seither wurden stetig neue globale sowie lokale Modelle entwickelt, deren Programmierung optimal auf die aktuelle Rechnerarchitektur zugeschnittenen ist. Somit konnte die Vorhersage sowohl qualitativ als auch bezüglich Vorhersagezeitraum optimiert werden. Wie in Abbildung 2.1 zu erkennen ist, wurde in den letzten 40 Jahren die Vorhersagbarkeit im Schnitt alle zehn Jahre um einen Tag verbessert. Dies bedeutet, dass heutzutage eine siebentägige Prognose die selbe Qualität aufweist wie eine viertägige Prognose von 1990 (Bauer et al., 2015; DWD, 2011). Ebenfalls in Abbildung 2.1 ist zu erkennen, dass es seit der Jahrtausendwende praktisch keinen Unterschied mehr gibt, ob die Vorhersage für die Nordhalbkugel oder die Südhalbkugel gilt. Der Grund dafür ist der vermehrte Gebrauch von Satellitendaten, die den grundsätzlichen Mangel an Daten auf der Südhalbkugel wett machen (Simmons & Hollingsworth, 2002).

ABBILDUNG 2.1: Verbesserung von Vorhersagen seit 1981



Güte (Forecast Skill) in Prozent für Drei-, Fünf-, Sieben-, und Zehntagesvorhersagen, aufgeteilt in nördliche Hemisphäre (NH) und südliche Hemisphäre (SH) (aus Bauer et al., 2015).

2.1.1 Globalmodelle

Numerische Wettermodelle bestehen grundsätzlich aus drei Teilen: Zu Beginn wird die Datenassimilation durchgeführt, bei der alle benötigten Daten von den verschiedenen Quellen gesammelt und ausgelesen werden. Danach folgt die eigentliche Kernberechnung, während der die nichtlinearen Gleichungssysteme gelöst werden und somit der Output generiert wird. Als Abschluss bei probabilistischen Modellen dient das stochastische Postprocessing, bei dem verschiedene Modellfehler korrigiert werden und der Output in ein für Menschen verständliches Format umgewandelt wird (Glahn & Lowry, 1972). Diese drei Teile werden im Folgenden näher erläutert. Sie variieren zwar in ihrer Zusammensetzung je nach Modellart, aber die Grundsätze gelten für alle Arten.

Datenassimilation

Bei numerischen Wettermodellen wird der Anfangszustand der Atmosphäre typischerweise aus einer Reihe von assimilierten Daten berechnet. Dabei werden Beobachtungsdaten mittels folgender Arten gewonnen (DWD, 2021b):

Bodenwetterstationen sind klassische Wetterstationen, beispielsweise auf Flughäfen oder an anderen geeigneten Orten. Sie verfügen über Messgeräte, die die Temperatur zwei Meter über Boden, den Bodendruck, die Feuchtigkeit und die Windgeschwindigkeit und -richtung messen.

Wetterballone steigen vom Boden durch die gesamte Troposphäre hoch und messen dabei konstant die Temperatur und den Taupunkt, den Luftdruck, den horizontalen Wind sowie manchmal den Ozonwert. Diese Daten werden direkt an eine Bodenstation gesendet.

Flugzeuge messen konstant die Temperatur, den statischen Druck und berechnen den Horizontalwind sowie die Turbulenz (Moninger et al., 2003). Ihre Bordcomputer brauchen diese für verschiedene Aufgaben wie Navigation und Autopilotfunktionen. Seit geraumer Zeit werden die gemessenen Parameter auch an Air Traffic Control-Stellen übermittelt. Wetterdienste können diese Daten ebenfalls auslesen mittels ACARS oder AMDAR. Trub et al. (2018) schlagen ausserdem ein Auslesen über ADS-B vor. Ein grosser Vorteil von Daten aus Flugzeugen ist, dass sie für die Wetterdienste relativ günstig zu akquirieren sind, da die Flugzeuge ohnehin fliegen (Petersen, 2016).

Windprofiler schicken elektromagnetische Wellen in leicht verschiedene Richtungen nach oben und messen das Rückstrahlsignal. Die Laufzeitmessungen ergeben durch den Dopplereffekt eine ganze Säule über dem Profiler, in der die Windrichtung und -stärke bekannt sind. Das Betreiben solcher Instrumente ist allerdings sehr teuer, weshalb sie nur selten benutzt werden (Bouttier, 2001). Gemäss Hocking et al. (2007) ist mittels Windprofilern auch eine Beobachtung der Ozonschicht möglich.

Bodengestützte Radarsysteme haben eine Vielzahl von Einsatzmöglichkeiten. Sie schicken beispielsweise ähnlich wie Windprofiler elektromagnetische Wellen aus und messen deren Rückstreuung, allerdings hauptsächlich von Wassertröpfchen in der Atmosphäre. Sie erzeugen somit ein Bild der vorhandenen Regenzellen (Battan, 1990; Doviak et al., 1979). Es lassen sich aber auch die Bewegung solcher Zellen messen, und aus dieser Bewegung kann wiederum die Windrichtung und -geschwindigkeit berechnet werden (Rinehart & Garvey, 1978).

Satelliten haben ebenfalls eine grosse Anzahl von Beobachtungsmöglichkeiten. Neben gewöhnlichen Bildern im für den Menschen sichtbaren Wellenspektrum zeichnen sie auch Infrarot- und Mikrowellenbilder auf (Neiburger & Wexler, 1961; Velden et al., 2003). Diese können aber nur die aufsummierte Menge an Strahlung messen, weshalb die 3D-Rekonstruktion der Atmosphäre nur näherungsweise möglich ist. Mit den klassischen Satellitenbildern kann auch die Bewegung von Wolken beobachtet werden. Durch diese sogenannten *Atmospheric Motion Vectors* lassen sich ebenfalls die vorhandene Windrichtung und -geschwindigkeit berechnen (Eyre et al., 2019). Eine weitere Möglichkeit der Atmosphärenbeobachtung entsteht, wenn ein Meteosatellit aus der Sicht eines anderen (typischerweise GNSS-) Satelliten hinter dem Horizont verschwindet. Der Meteosatellit empfängt das GNSS-Signal, das das Profil der Atmosphäre durchlaufen hat, und kann deren Brechungsindex auswerten. Dieser gibt wiederum Auskunft über die integrierte Temperatur, Feuchte und Dichte der Atmosphäre (Eyre et al., 2021).

Alle diese Messdaten haben aber einen grossen Schwachpunkt: Ihre Verteilung ist sehr inhomogen, weshalb der gesamte Atmosphärenzustand allein durch die Assimilation dieser Daten ungenügend beschrieben wird. In den 1950er-Jahren, zu Beginn der Datenassimilation, wurde aus den gemessenen Werten der Rest der Atmosphäre mittels Interpolation geschätzt. Später wurden dafür dreidimensionale Methoden inklusive multivariater Zusammenhänge benutzt (Lorenc, 1981). Heutzutage gehören vierdimensionale Methoden zum Standard. Dabei werden die zusammengetragenen Daten mittels verschiedener Filtertechnologien aufbereitet und mit dem Vorhersagemodell der vorherigen Rechenperiode abgeglichen, um ein kompletteres Abbild der Atmosphäre zu erstellen (Gustafsson et al., 2018).

Modellberechnung

Sobald der Anfangszustand der Atmosphäre bekannt ist, kann die Modellberechnung beginnen. Auch hier gab es seit Beginn der numerischen Wettervorhersage signifikante Verbesserungen bezüglich der Modellarchitekturen. Um eine Berechnung durchzuführen, wird die Atmosphäre in ein Gitter eingeteilt.

Die offensichtlichste Methode dafür ist die Einteilung in Längen- und Breitengrade, wie sie beispielsweise auch in der Geografie verwendet wird. In der Höhe wird die Atmosphäre ebenfalls in verschieden grosse Abschnitte unterteilt. Dadurch entstehen Kuben, wie beispielsweise beim früher weit verbreiteten Arakawa-C-Gitter (Abbildung 2.2). Dabei wird für die Parameter Druck, Temperatur und Feuchtigkeit jeweils einfach ein Wert im Zentrum des Kubus verwendet. Der Wind in u-, v- und w-Richtung wird in der Mitte der jeweiligen Kanten positioniert. Dies erlaubt relativ einfache Berechnungen beispielsweise bezüglich Advektion (Anand & Moorhead, 1995). Diese Art von Einteilung weist aber grosse Probleme an den Polen auf, da eine Einteilung in Kuben dort nur sehr schwer machbar ist (Bénard & Glinton, 2019; Buckeridge & Scheichl, 2010; COMET, 2015).

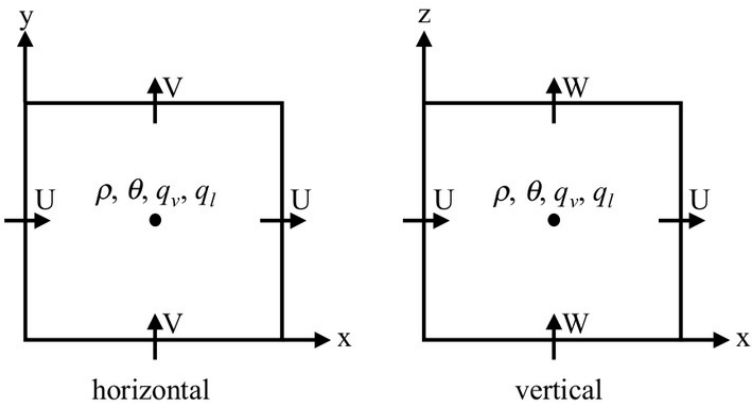


ABBILDUNG 2.2: Arakawa-C-Gitter (aus ResearchGate, 2021).

Die Abhängigkeiten zwischen den Kuben werden unter anderem mit Legendre-Transformationen beschrieben. Diese haben den Nachteil, dass sie sich sehr schnell vergrössern (dritte Potenz), wenn über viele Zeitschritte gerechnet wird (Satoh et al., 2008). Je weiter also die Vorhersage in die Zukunft reicht, desto grösser wird der potenzielle Fehler.

Eine weitere Möglichkeit zur Einteilung der Atmosphäre ist das Spektral- oder Strahlenmodell. Als Output werden die Daten dabei als Wellenfunktion dargestellt, beispielsweise die Höhe einer 500 hPa Linie um die Welt. Diese Art von Modellen ist aber in ihrer maximalen Auflösung begrenzt durch die maximale Anzahl Wellenfunktionen, sie ist also nicht beliebig skalierbar (COMET, 2015).

Um die verschiedenen Probleme der beschriebenen Modelle zu lösen, wurde seit Ende des letzten Jahrtausends an einem Gittermodell geforscht, das durch seine ikosaedrale Struktur charakterisiert wird (Majewski et

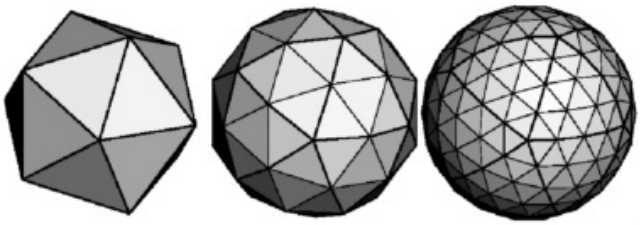


ABBILDUNG 2.3: Gittermodell mit ikosaedraler Struktur (aus Oeser, 2021).

al., 2002). Dieses orientiert sich nicht mehr an den klassischen Breiten- und Längengraden, sondern stellt die Welt einem Ikosaeder dar, dessen Kantenlängen dann beliebig verfeinert werden können (Abbildung 2.3). Die Felder über dem Globus sind nun also nicht mehr rechteckig, sondern dreieckig. Dies hat den Vorteil, dass alle Seitenlängen überall gleich lang sind, auch an den Polen (Bénard & Glinton, 2019; Chaudhari et al., 2007).

Tomita und Satoh (2004) konnten zeigen, dass mit dieser Art von Gitter eine viel höhere Auflösung möglich ist, während die Rechenleistung in etwa gleich bleibt, verglichen mit rechteckigen Modellen. Ausserdem vereinfacht diese Herangehensweise die nonhydrostatische Berechnung, was die Simulation von Wolken erheblich verbessert (Satoh et al., 2008). Die ikosaedrale Gittersruktur stellt heute den neusten Stand der Technik dar und wird bereits in numerischen Wettermodellen verwendet, beispielsweise beim *Icosahedral Nonhydrostatic Model* ICON des Deutschen Wetterdienstes DWD (2021c), beschrieben durch Zängl et al. (2015).

Postprocessing

Sobald eine Modellberechnung erstellt worden ist, kann ein sogenanntes Postprocessing durchgeführt werden. Hierbei werden beispielsweise verschiedene Produkte wie Graphiken und Bilder erstellt. Eine weitere Art von Postprocessing ist das sogenannte Model Output Statistics (MOS), bei dem die vom Modell ausgegebenen Werte an lokale Gegebenheiten angepasst werden. Die Erfassung dieser Gegebenheiten basieren auf langjährigen Korrelationsrechnungen zwischen Wettermessdaten und -Prognosen. Somit können kleinste orographische Prozesse im Bereich weniger Kilometer wie beispielsweise Kaltluftseen vorhergesagt werden (Glahn & Lowry, 1972; Gneiting et al., 2005). Diese linearen Methoden sind sehr bewährt, können aber auch eher aufwendig anzupassen sein. In den letzten Jahren ist deshalb die Diskussion entstanden, für das Postprocessing künstliche Intelligenz und machine learning zu verwenden. Operationell wird dies allerdings noch nirgends gemacht (Haupt et al., 2021).

2.1.2 Ensemblemodelle

Seit über 50 Jahren ist bekannt, dass die Atmosphäre ein chaotisches System ist. Der Meteorologe Edward Lorenz erlangte 1972 mit seiner Schmetterlingstheorie weltweite Berühmtheit. Der DWD (2011) schreibt dazu in seinem Magazin *promet*:

> *„Lorenz zeigt in dieser Publikation zum ersten Mal, dass einfache deterministische Systeme prinzipiell nur für einen endlichen Zeitraum vorhersagbar sind, wobei diese Zeitspanne von der Genauigkeit abhängt, mit der die Anfangsbedingungen spezifiziert werden. Dies bedeutet einerseits, dass der mögliche Vorhersagehorizont größer wird, je genauer wir die Anfangsbedingungen kennen. Andererseits müssen wir beachten, dass selbst bei immer kleiner werdenden Fehlern in den Anfangsbedingungen dennoch immer eine Restunsicherheit bestehen bleibt, und es damit immer eine Grenze der Vorhersagbarkeit im Sinne der Anfangsbedingungen geben wird.“*

Um dieses Problem der deterministischen Wettervorhersage zu entschärft, entwickelten die Wetterdienste einen probabilistischen Ansatz. Sie erstellten auf Basis ihrer bestehenden Wettermodelle sogenannte Ensemblemodelle. Bei diesen wird nicht nur ein Modell von Anfang bis Ende durchgerechnet, sondern eine Vielzahl

gleichzeitig. Typischerweise handelt es sich hier um 5 bis 100 sogenannter Ensemblemember (Gneiting & Raftery, 2005). Jedes dieser Ensemblemember hat leicht andere Anfangsbedingungen, was schliesslich auch einen anderen Output generiert. Ebenfalls werden bei einigen dieser Member bestimmte Parameter leicht verändert, was somit die Werte der zu lösenden Gleichungen verändert. Theis et al. (2015) beispielsweise beschreiben für das ehemalige lokale Ensemblemodell des DWD, dem COSMO-DE-EPS, wie als Anfangsbedingungen vier verschiedene Globalmodelle verwendet werden. Jedes dieser vier Modelle wurde fünf Mal verwendet, jeweils mit leicht anderen Parametern. Änderungen gab es im Bereich der Konvektionsberechnung, der Wolkenmikrophysik oder der Turbulenz. Somit entstanden 20 leicht unterschiedliche Modelle. Jedes dieser Modelle berechnete die atmosphärischen Zustände leicht anders. Somit konnte nach dem oben beschriebenen Postprocessing eine Wahrscheinlichkeit für ein eintretendes Ereignis angegeben werden.

Heutzutage hat praktisch jedes globale sowie lokale Wettermodell seinen eigenen Ensemble-Zwilling.

2.1.3 Lokalmodelle

Der wohl grösste Nachteil aller Globalmodelle ist, dass die Atmosphäre unseres Planeten so gross und komplex ist. Um sie in ihrem gesamten Umfang zu berechnen und abzubilden, muss gezwungenermassen eine Vereinfachung stattfinden. Denn obwohl die Rechenleistung von Supercomputern stetig zunimmt, ist ihre Kapazität nach wie vor begrenzt. Aus diesem Grund muss bei Globalmodellen die Gitterweite relativ gross gewählt werden (Schär et al., 2020). Dies bedeutet, dass viele kleinskalige Prozesse wie beispielsweise die Konvektion oder orographische Gegebenheiten parametrisiert werden müssen. Diese Prozesse sind nämlich so lokal, dass sie bei Globalmodellen innerhalb nur einer einzigen Gitterzelle entstehen würden (Bauer et al., 2015). Durch deren Parametrisierung kann weder deren Lebensdauer noch ihre Interaktion mit den globalen Geschehnissen optimal beschrieben werden. Dies ist jedoch für die lokale Wetterprognose sehr wichtig.

Aus diesem Grund entwickelten die meisten meteorologischen Dienste Lokalmodelle für ihr jeweiliges Kerngebiet. Die Grundarchitektur wird dabei meist von den Globalmodellen übernommen, wobei die Auflösung des Gitters viel feiner ist. Dies ist aber nur für ein begrenztes geographisches Gebiet möglich. Deshalb braucht es als Rahmenbedingung immer ein grösseres Lokalmodell oder ein Globalmodell, denn sonst würde der berechnende Computer an den Rändern einfach aufhören zu rechnen (Baldauf et al., 2011). Somit können aber auch lokale Gewitter oder landesspezifische Spezialitäten wie der Einfluss von einzelnen Bergen simuliert werden. Engmaschige Lokalmodelle sind daher auch oft konvektionsauflösende Modelle.

Ein Nachteil vieler Lokalmodelle ist, dass sie in der Entwicklung zum Teil um Jahre hinter den Globalmodellen hinterherhinken (Gustafsson et al., 2018). Ein Grund dafür liegt insbesondere bei kleineren meteorologischen Diensten in den begrenzten Ressourcen (Folini, 2018).

2.1.4 ICON und COSMO-Modelle

Das Consortium for Small-scale Modelling COSMO (2022) ist ein 1998 gegründeter Zusammenschluss des Deutschen Wetterdiensts DWD mit MeteoSchweiz. Weitere Mitglieder sind die nationalen Wetterdienste von Italien, Griechenland, Polen, Rumänien, Russland und Israel. Sie werden unterstützt von verschiedenen regionalen sowie militärischen Institutionen aus Deutschland und Italien. Ziel des Konsortiums

ist es, Erfahrungen zu kombinieren und dadurch ein numerisches Wettermodell, bestehend aus globalen und lokalen Submodellen, zu entwickeln und aufrechtzuerhalten. Die daraus entstandenen COSMO-Modelle sind heute teilweise noch immer im Einsatz. Bei MeteoSchweiz (2021) sind dies beispielsweise das COSMO-2 und das COSMO-7.

Seit Mitte des letzten Jahrzehnts hat der DWD ein Nachfolgemodell entwickelt, das *Icosahedral Nonhydrostatic Model* ICON. Im Januar 2015 löste dieses in Deutschland das COSMO-Modell als globales Modell ab. Im Dezember 2016 folgte die Umstellung des Lokalmodells COSMO-EU auf das ICON-EU, und im Januar 2021 wurde das COSMO-D2 durch das ICON-D2 ersetzt (Rieger, 2021). Ziel ist es, dass auch die anderen Staaten des COSMO-Konsortiums bis 2022 die ICON-Modelle bei sich implementieren (COSMO, 2022). Denn während das COSMO gerade bei steilen orographischen Verhältnissen, wie sie im Alpenraum häufig vorkommen, noch Mühe bekundete (Brdar et al., 2012), verspricht das ICON, diese besser zu modellieren. Weitere Vorteile des ICONs sind eine verbesserte Programmierung, welche weniger Rechenressourcen bei gleichbleibender bis höherer Vorhersagequalität benötigt. Ausserdem bietet das ICON die Möglichkeit, die Gittergrösse praktisch beliebig zu verfeinern (Rieger, 2021).

Das ICON-Basismodell umspannt den gesamten Globus mit einem Arakawa-C-Gitter in ikosaedraler Form (Abbildungen 2.2 und 2.3). Die Maschengrösse beträgt 13 km (Zängl et al., 2015). In der Höhe besteht das Modell aus 90 Ebenen. Diese folgen im unteren Teil dem Gelände und liegen nahe beieinander. Je höher die Ebenen liegen, desto flacher und weiter auseinander sind sie. Die oberste Ebene befindet sich auf einer Höhe von 75 km. Das Modell wird alle sechs Stunden neu berechnet (DWD, 2021c).

Das Lokalmodell ICON-EU verfügt über eine Maschengrösse von 6.5 km und erstreckt sich über ganz Europa (Rieger, 2021). Die Höhenarchitektur ist analog zum Globalmodell, wobei die obersten 30 Ebenen weggelassen werden. Es besteht also aus 60 Ebenen. Als Rahmenbedingung dient das Basismodell ICON, die Berechnung wird alle drei Stunden durchgeführt. Auf dem Open-Data-Server des DWD (2020) sind die berechneten Daten frei verfügbar. Sie sind jeweils pro Modellebene in einer grib2-Datei gespeichert.

Das konvektionsauflösende ICON-D2-Modell hat eine Maschengrösse von zwei Kilometern. Es deckt alle deutschsprachigen Länder sowie die Benelux-Staaten und Dänemark und Teile derer Nachbarländer ab. Es besteht aus 66 Ebenen bis zu einer Höhe von 22 km. Das ICON-D2 benutzt eine Kombination aus ICON-EU und dem Integrated Forecast System IFS des ECMWF als Rahmenbedingung. Die Berechnung und Speicherung funktionieren analog zum ICON-EU (Reinert et al., 2021).

2.2 Softwarepakete und Dateiformate

2.2.1 Grib2

Die in diesem Projekt verwendeten Modelldateien sind im Dateiformat grib2 gespeichert. Grib2 ist die Weiterentwicklung des Dateiformats grib. Der Name ist eine Abkürzung und steht für *General Regularly-distributed Information in Binary form*. Das Format wurde vom World Meteorological Office WMO für den internationalen Austausch grosser Datenmengen entwickelt. Es eignet sich sehr gut zur Speicherung von Vorhersagedaten, da alle nötigen Metadaten in der Datei mitgegeben werden. Eine

grib2-Datei besteht aus acht Sektionen, wobei in den ersten sechs Sektionen die Metadaten gespeichert sind. In Sektion sieben sind die tatsächlichen Daten gespeichert, und Sektion acht bildet den Abschluss in Form der Zahlenreihe 7777. Die Sektionen zwei bis sieben können sich wiederholen, was dem Format im Gegensatz zu seinem Vorgänger grib Version 1 unter anderem ermöglicht, Ensemblemodelle zu speichern (Schröder, 2014). Ein weiterer Vorteil von Version 2 ist, dass sich damit auch ikosaedrale Strukturen speichern lassen, wie sie beispielsweise im ICON-Modell des DWD (2021c) verwendet werden.

2.2.2 ecCodes

Die in diesem Projekt verwendeten grib2-Dateien werden mit dem Softwarepaket ecCodes verarbeitet. ecCodes ist eine Weiterentwicklung des Pakets GRIB-API; beide wurden vom European Centre for Medium-Range Weather Forecasts ECMWF entwickelt. ecCodes enthält Interfaces für die Programmiersprachen C, Fortran 90 und Python 3 (Abellan, 2021). Mit den ecCodes-Befehlen können grib2-Dateien abgeändert, ausgelesen oder kopiert werden.

2.3 LNAS

Die Zivilluftfahrt ist seit ihrer Entstehung vor über hundert Jahren stetig gewachsen. Zwar gab es auch immer wieder Krisen in der Luftfahrt, während derer die Nachfrage nach Passagierflügen kurzfristig zusammengebrochen ist. Jedoch hat sich die Branche danach immer wieder erholt und neue Rekordzahlen aufgestellt. Insgesamt ist sie so global um durchschnittlich fünf Prozent pro Jahr gewachsen (Airbus, 2021). Auch wenn die aktuelle Covid-Pandemie die bisher grösste Krise der Zivilluftfahrt darstellt, kann davon ausgegangen werden, dass sich langfristig die Nachfrage wieder erholen und das Vorkrisenniveau sogar übersteigen wird. Gemäss EASA und EAA (2019) und ihrem European Aviation Environmental Report wird eine Zunahme der weltweiten Flüge von 42 % gegenüber 2019 erwartet. Gleichzeitig wächst die Bevölkerung ebenfalls, vor allem in Ballungszentren und Agglomerationen; also dort, wo häufig auch Flughäfen angesiedelt sind. Diese beiden Faktoren führen unter anderem dazu, dass immer mehr Menschen den Effekt von Flugzeugemmissionen in Form von Lärm und Schadstoffen spüren (Griefahn et al., 2006). Deshalb muss versucht werden, diese Emissionen zu verringern.

Die effektivste Methode dafür ist der Einsatz leiserer Flugzeuge. Dies wird von den Airlines bereits praktiziert. Da jedoch moderne Verkehrsflugzeuge eine grosse Nutzungsdauer haben, dauert es lange, bis sie ersetzt werden. Ausserdem dauert die Entwicklung neuer Flugzeuge mehrere Jahre und der Kauf solcher ist sehr teuer. Deshalb müssen auch kurzfristige Lösungen in Betracht gezogen werden.

Eine Möglichkeit dabei ist, die Start- und Landeverfahren anzupassen. Beim Landeanflug beispielsweise kann darauf geachtet werden, dass die Hochauftriebshilfen (Landeklappen) und das Fahrwerk möglichst spät ausgefahren werden. Diese erzeugen nämlich im ausgefahrenen Zustand sehr viel Lärm (Bauer & König, 2016). Andererseits muss ein Verkehrsflugzeug spätestens bei einer Höhe von 1000 Fuss über der Piste die vollständige Landekonfiguration erstellt sein und die Endanfluggeschwindigkeit erreicht haben. Man spricht dann von einem stabilisierten Anflug. Gerade die Landung ist für Pilot:innen aber eine sehr intensive Flugphase, weshalb sie oft Sicherheitsmargen einbauen, damit eine sichere Landung gelingt und sie nicht

durchstarten müssen. Dies führt dazu, dass die Landeklappen und das Fahrwerk früher als nötig ausgefahren werden. Dies benötigt jedoch eine höhere Triebwerkleistung, was wiederum eine höhere Lärmemission erzeugt (Abdelmoula & Scholz, 2018).

Um die Pilot:innen im Landeanflug dabei zu unterstützen, die Landeklappen und das Fahrwerk zum optimalen Zeitpunkt auszufahren, haben Kühne et al. (2018) vom Deutschen Zentrum für Luft- und Raumfahrt DLR ein Assistenzsystem namens Low Noise Augmentation System (LNAS) entwickelt. Dieses System zeigt den Pilot:innen während dem Anflug an, wann der optimale Zeitpunkt dafür ist, die Klappen- und Fahrwerksschalter zu betätigen. Auf dem Bildschirm ihres Electronic Flightbags (EFB) sehen sie, wie lange es noch dauert, bis sie den nächsten Konfigurationsschritt zu tätigen haben. Ausserdem wird angezeigt, wann der optimale Zeitpunkt ist, die Geschwindigkeit zu reduzieren. Diese Funktionen tragen dazu bei, dass möglichst wenig Triebwerkschub benötigt wird. Das LNAS hilft also auch, Treibstoff zu sparen. Die Entwickler konnten zeigen, dass das System ausserdem zu einer Lärmverminderung von bis zu zwei Dezibel gegenüber gewöhnlichen Anflügen führen kann (Abdelmoula & Scholz, 2018; Abdelmoula et al., 2017).

Seit der ersten Entwicklung des LNAS 2016 wurde das System kontinuierlich weiterentwickelt sowie ausgiebig getestet. Dies geschah sowohl mit dem DLR- eigenen Testflugzeug ATRA (Advanced Technology Research Aircraft) als auch im Linienbetrieb bei Lufthansa. Tests fanden unter anderem an den Flughäfen Frankfurt und Zürich statt, unter realen Flugbedingungen bezüglich anderem Verkehr und Kommunikation mit Fluglotsen (Jäger et al., 2021). Flugversuche im regulären Liniendienst bei den Fluggesellschaften Lufthansa und SWISS sind in naher Zukunft geplant.

Für die interne Berechnung im LNAS spielen die Wetterbedingungen während des Anflugs eine zentrale Rolle. Parameter wie Windrichtung und -geschwindigkeit, aber auch Luftdruck, Temperatur und Feuchtigkeit, haben einen grossen Einfluss auf die optimalen Zeitpunkte zur Konfiguration. Daher ist es wichtig, dem LNAS diese Parameter so genau wie möglich zur Verfügung zu stellen. Bei der Lufthansa werden deshalb diese Wetterdaten von einem vorausfliegenden Flugzeug direkt zum nächsten geschickt. Somit stehen sehr aktuelle und genaue Daten zur Verfügung. Da die A320-Flotte der SWISS aber nicht über Inflight-Connectivity verfügt, ist diese Funktionsweise bei ihr nicht möglich. Aus diesem Grund werden lediglich die von der Bodenstation auf dem Flughafen gemessenen Daten eingegeben. Zusätzlich werden die vom Flugzeug gemessenen Daten verwendet. Um den Zustand der Atmosphäre dazwischen zu berechnen, wird linear interpoliert. Dies kann unter Umständen zu einer erheblichen Verzerrung führen. Ziel dieser Arbeit ist deshalb, eine Applikation zu entwickeln, mithilfe derer Prognosedaten der Atmosphäre vor dem Start der Flugzeuge in das LNAS geladen werden können.

Kapitel 3

Methodik

3.1 Umgebung und Programme

Ziel dieser Arbeit war, aktuelle Vorhersagedaten in einem dynamisch anwendbaren Format zur Verfügung zu stellen. Um dieses zu erreichen, wurde zu Beginn eine Evaluation durchgeführt, welche Daten benötigt würden. Als Input wurden dem Autor vom Betreuer die ICON-Modelle des Deutschen Wetterdienstes DWD empfohlen.

3.1.1 Windows oder Linux

Der Autor arbeitet standardmässig mit dem Betriebssystem Microsoft Windows. Damit wurde zunächst ein einzelnes File des ICON-D2-Modells manuell vom Open-Data-Servers des DWD (2020) heruntergeladen und im Ordner *Downloads* gespeichert. Die Dateien werden standardgemäss mit der Endung grib2.bz2 gespeichert. Das Dateiformat grib2 wird im Abschnitt 2.2.1 beschrieben, und die Komprimierungsart bz2 wird im Abschnitt 3.1.3 erläutert.

Eine entpackte grib2-Datei kann gemäss Schröder (2014) entweder mit dem Programm wgrib2 der NOAA oder mit dem Programm GRIB-API des ECMWF verarbeitet werden, wobei er letzteres für Dateien des DWD empfiehlt. Da GRIB-API aber seither nicht mehr operationell ist, wird als Nachfolgeprogramm ecCodes (siehe Abschnitt 3.1.3) angegeben (Fucile, 2021). ecCodes wird von den Entwicklern als Programm für Linux empfohlen. Eine Version für Windows bestehe zwar, aber es kann keine korrekte Funktionsweise garantiert werden (Santoalla, 2021). Aus diesem Grund wurde entschieden, dieses Projekt unter Linux zu entwickeln. Eine einfache Möglichkeit, innerhalb der Windowsumgebung eine virtuelle Linuxumgebung zu erstellen, bietet Windows Subsystem for Linux (WSL, 2021).

3.1.2 Matlab oder Python

Ursprünglich war die Idee, die kommerzielle Software Matlab von MathWorks (2021) zu verwenden, da ein solches Skript gemäss dem Zweitbetreuer optimal in das LNAS-System integriert werden könne. Andererseits stellt ecCodes eine eigene Library für die frei verfügbare Programmiersprache Python (2021) zur Verfügung, in welcher alle Befehle zum Verarbeiten der grib2-Dateien enthalten sind. Dieser Vorteil für die Entwicklung wurde vom Autor als höher gewichtet, weshalb das Skript mittels Python entwickelt wurde. Diese Entscheidung ist auch im Sinne von Open Science gefallen.

3.1.3 Weitere benötigte Software

In diesem Abschnitt werden zusätzliche Programme und Befehle vorgestellt, welche in der Entwicklung dieses Projekts verwendet wurden.

ecCodes: Mit Linux unter WSL (Ubuntu) konnte das im Abschnitt 2.2.2 beschriebene Programm ecCodes installiert werden. Da die Anforderung war, für bestimmte Koordinaten die Werte zu suchen, wurde aus der ecCodes-Library der Befehl `codes_grib_find_nearest` verwendet. Bei diesem werden Koordinaten im Latitude-/Longitude-Format angegeben, und als Output werden neben dem Wert auch der Index und die Distanz für den Gitterpunkt, der den Koordinaten am nächsten liegt, ausgegeben. Ausserdem wird der Befehl `codes_get_values` verwendet. Er gibt den Wert eines Feldes an einem angegebenen Index aus. Der Output des ersten Befehls kann also direkt für den zweiten als Input verwendet werden.

bunzip2: Wie im Abschnitt 3.1.1 beschrieben wird, sind die verwendeten Dateien im bz2-Format komprimiert. Eine bz2-Datei wird gemäss Seward und Blättermann (2019) mit dem Commandlinebefehl `bunzip2` entpackt. Es besteht ebenfalls eine Python-Library, welche in das Skript importiert wird.

PyCharm: Für die Entwicklung des Pythonskripts wurde die Entwicklerumgebung (IDE) PyCharm von JetBrains (2022) benutzt, welche dem Autor bereits aus früheren Arbeiten bekannt war. Da das Softwarepaket ecCodes unter Linux verwendet wurde (vgl. Abschnitt 3.1.1), musste auch im PyCharm der Python Interpreter auf Linux respektive WSL umgeschaltet werden. Dies ist in der frei verfügbaren PyCharm Community Edition jedoch nicht möglich, weshalb auf die zahlungspflichtige Professional-Version zugegriffen werden musste. Diese Abweichung von der Open Science Philosophie ist hier unvermeidlich. Die aktuellste Version von PyCharm Professional war zum Zeitpunkt der Entwicklung die Version 2021.2. Diese enthielt zu jener Zeit jedoch einen Bug, durch den kein WSL Interpreter ausgewählt werden konnte. Deshalb wurde die ältere Version 2021.1 benutzt, zusammen mit einem Workaround von Karateev und Karpov (2021).

Git und GitHub: Zur Versionenkontrolle und zur Sicherung der geleisteten Arbeit wurde Git verwendet. Git ist in PyCharm bereits integriert und lässt sich in wenigen Schritten einrichten (JetBrains, 2021a). Der Vorteil eines Dateiversionierungsprogramms ist, dass sich Änderungen am Code gut überblicken lassen. Ausserdem kann notfalls auf eine ältere Version des Codes zurückgegriffen werden, falls beim Entwickeln ein grober Fehler geschieht, den man nicht mehr von Hand rückgängig machen kann. Die Stärke von Git ist, dass eine Kopie der Versionenkontrolle online auf GitHub (2022) gespeichert werden kann. Von dort kann ein Projekt mit anderen Usern geteilt werden. Dieses Projekt wurde ebenfalls auf GitHub gespeichert, jedoch im Privatmodus. Dies bedeutet, dass nur vom Entwickler zugelassene Personen Zugriff auf den Quellcode haben.

3.2 Entwicklung

In diesem Abschnitt werden die jeweiligen Arbeitsschritte beschrieben, die für die Entwicklung des Python-Skripts getätigt wurden.

Erstes Ziel war es, eine Funktion in Python zu erstellen, die automatisch eine einzelne Datei vom Open-Data-Servers des DWD (2020) herunterlädt. Dazu musste eine URL zusammengestellt werden. Die Benennungskonvention des Dateinamens ist standardisiert; die URL besteht aus einem fixen und einem variablen Teil. Letzterer wird in Abbildung 3.1 beschrieben. Er besteht aus der Angabe des Modellnamens und der Region (braun), des Modellgitters (violett), des Initialisierungszeitpunktes

https://opendata.dwd.de/weather/nwp/icon-d2/grib/09/u/icon-d2_germany_regular-lat-lon_model-level_2022011109_037_6_u.grib2.bz2

ABBILDUNG 3.1: Beispiel eines Links für eine ICON-D2 Datei.

(grün), des Vorhersagezeitpunktes (blau), der Modellebene (orange) sowie der Variable (rot). Für den ersten Downloadversuch wurden die variablen Teile der URL als String im Skript festgelegt.

Der zweite Entwicklungsschritt war daher eine Funktion, die die aktuelle Zeit ausliest und sie in einen String umwandelt, dessen Format sich mit der URL deckt. Somit konnte das Skript zu jeder beliebigen Zeit den aktuellsten Modelllauf herunterladen. Diese Zeitfunktion wurde so konstruiert, dass auch die Vorhersagestunde mit herausgegeben wird, welche zur aktuellen Zeit passt. Wenn also beispielsweise um 12 Uhr eine Abfrage stattfindet, und der aktuellste Modelllauf um 9 Uhr begonnen hat, wird als Vorhersagestunde eine 3 in die URL gebaut. Ausserdem folgte auch eine Funktion, die die heruntergeladenen Dateien entpackt. Siehe dazu auch Abschnitt 3.1.3.

Wie in Abschnitt 2.1.4 beschrieben, sind die ICON-Modelle pro Modellebene jeweils in einer eigenen Datei gespeichert, und diese Ebenen befinden sich je nach Gelände nicht immer auf der gleichen geometrischen Höhe. Daher wird eine Methode benötigt, die für die jeweils angegebenen Koordinaten aus der angegebenen geometrischen Höhe die Modellebene bestimmt. Das DWD stellt auf seinem Open-Data-Server auch zeitunabhängige HHL-files zur Verfügung. HHL steht für *geometric Height of Half Level*, zu Deutsch geometrische Höhe zwischen zwei Modellebenen. Dies entspricht der Grenze von einer zur nächsten Ebene. In den HHL-Dateien ist für jeden Gitterpunkt diese Höhe in Meter angegeben. Der Mittelpunkt zwischen zwei Höhen entspricht der Höhe einer Modellebene. Mit dieser Information kann nun anhand der angegebenen Höhe die benötigte Ebene bestimmt werden.

An diesem Punkt waren die Basisfunktionen des Skripts, eine aktuelle Vorhersagedatei herunterzuladen, zu entpacken und den Wert an der angegebenen Stelle auszugeben, erreicht. Dies bedeutete, dass das Skript ausgebaut und weiterentwickelt werden konnte. Folgende Funktionen wurden hinzugefügt. Ihre Arbeitsweise wird im Abschnitt 4.1 beschrieben.

- Es wurde ein for-loop hinzugefügt, der über die Variablen Druck p, Temperatur t, relative Feuchte qv und die drei Windrichtungen u, v und w iteriert.
- Die Zeitfunktion wurde so angepasst, dass auch Vorhersagestunden in der Zukunft ausgegeben werden können. Dies im Gegensatz dazu, dass zuvor nur Daten der aktuellen Zeit ausgegeben wurden.
- Eine Funktion wurde eingebaut, die überprüft, ob der aktuellste Modelllauf bereits verfügbar ist. Falls dies nicht der Fall ist, wird der nächstältere heruntergeladen.
- Als weiteres Feature wurde ein ICON-Switcher hinzugefügt, der zwischen dem ICON-D2- und dem ICON-EU-Modell wechselt. Wie im Kapitel 2.1.4 beschrieben, sind diese in ihrem geographischen Umfang verschieden. Daher überprüft der ICON-Switcher anhand der angegebenen Koordinaten, welches der beiden Modelle passt, und wählt dieses dann aus.

- Es wurde eine Funktion programmiert, welche alle Dateien, die vor mehr als vier Stunden heruntergeladen wurden, wieder löscht. Diese optional verwendbare Funktion ist für Anwender gedacht, deren verfügbarer Speicherplatz begrenzt ist.
- Eine Funktion wurde hinzugefügt, die den Output des Skripts in eine csv-Datei schreibt.

3.3 Performance

Nach dem Hinzufügen und Ausbauen der verschiedenen Funktionen wurde die Performance des Skripts analysiert. In einem ersten Schritt geschah dies mit mehreren Zeitausgaben, welche in den Code integriert wurden. An mehreren Stellen im Code wurden Printbefehle eingefügt, welche die aktuelle Zeit mit einer Genauigkeit von einer Millisekunde angeben. Mit dieser sehr aufwendigen Methode liess sich grob überblicken, wie lange das Skript insgesamt und die einzelnen Funktionen für sich jeweils dauern. Dabei wurden vor allem zwei Aspekte als Flaschenhals identifiziert: Das Herunterladen der Dateien aus dem Internet und der ecCodes-Befehl `codes_grib_find_nearest`.

Als Alternative zum unflexiblen Analysieren mittels Zeitausgaben wurde dem Autor vom Betreuer Profiler empfohlen. Die verwendete Entwicklerumgebung Pycharm hat bereits einen Profiler integriert (JetBrains, 2021b). Ein Ausschnitt einer mittels Profiler erzeugten Graphik ist in Abbildung 3.2 zu sehen. Die Analyse der Codes mit dem Profiler bestätigte die beiden zuvor gefundenen Flaschenhälse. Dabei wurde erkannt, dass beim ersten hauptsächlich die Abfrage der URL auf dem Server lange dauert. Beim eweiten ist das Problem vor allem, dass der Befehl während eines Durchgangs bis zu 2400-mal benutzt wird.

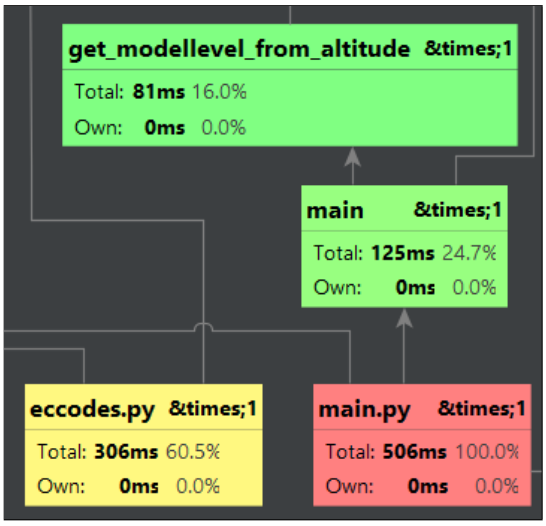


ABBILDUNG 3.2: Ausschnitt einer Analyse mit dem PyCharm-Profiler.

Um das Herunterladen zu beschleunigen, wurde eine Funktion eingebaut, die überprüft, ob eine benötigte Datei allenfalls bereits heruntergeladen wurde. Falls ja, wird sie nicht nochmals heruntergeladen. Somit können unnötige Downloads vermieden werden, was die Ausführung des Skripts beschleunigt.

Als weiterer Verbesserungsschritt wurde nach einer Möglichkeit gesucht, die Anzahl der Benutzung des Befehls `codes_grib_find_nearest` zu verringern. Der Code wurde so abgeändert, dass der Befehl nur noch bei einer HHL-Datei verwendet wurde. Gleichzeitig wird bei dieser Verwendung nur noch der Index des Gitterpunkts herausgelesen, der den angegebenen Koordinaten am nächsten liegt. Dieser Index hat in allen grib2-Dateien für einen bestimmten Koordinatenpunkt denselben Wert. Dies ermöglicht beim Auslesen der Variabeldateien die Benutzung des massiv schnelleren ecCodes-Befehls `codes_get_values`, der den Index als Input benötigt. Da jeweils sechs Variablen und 61 resp. 66 HHL-Dateien abgefragt werden, konnte

mit dieser Umstellung die Anzahl des nearest-Befehls um Faktor 67 resp. 72 reduziert werden.

Die Resultate dieser Performanceverbesserungen werden im Kapitel 4.3 beschrieben.

3.4 Validierung

Das in dieser Arbeit entwickelte Skript liest Vorhersagedaten der ICON-Modelle des DWD aus und stellt sie dem Benutzer für spezifische Punkte zur Verfügung. Um zu überprüfen, ob die ausgegebenen Daten erstens realistisch sind, und zweitens einen Mehrwert für das LNAS-System darstellen, muss eine Validierung durchgeführt werden.

Für diese Validierung wurden vom Industriepartner SWISS mehrere Messdatensätze bereitgestellt. Diese anonymisierten Daten wurden von Flugzeugen der SWISS-A320-Flotte gemessen, und zwar im Anflug auf den Flughafen Zürich ab einer Flughöhe von 10'000 Fuss bis zur Landung auf 1410 Fuss. Es handelte sich um sechs reguläre Passagierflüge, welche am Nachmittag des 17. Januar 2022 zwischen 15 Uhr und 19 Uhr Lokalzeit gelandet sind. Die erhobenen Parameter sind die Uhrzeit in UTC, die Koordinaten im Latitude-/Longitude-Format, die geometrische Höhe über Meer in Fuss, der statische, der dynamische und der totale Druck in Millibar, die Aussentemperatur in Grad Celsius, die Windrichtung in Grad geographisch sowie die Windgeschwindigkeit in Knoten. Pro Flug wurden zwischen 400 und 550 Datenpunkte geliefert, jeweils in einer Textdatei im kommaseparierten Format.

Um die gemessenen Daten mit Vorhersagen zu vergleichen, mussten die Flüge im Python-Skript rekonstruiert werden. Dabei wurden die Koordinaten, die Höhe und die Uhrzeit der Flüge in das Skript eingegeben und das Skript gestartet. Dies wurde für alle Flüge zu verschiedenen Zeitpunkten durchgeführt, zwischen einer und sechs Stunden vor der Landung. Somit konnte simuliert werden, dass die Daten vor dem Start eines Fluges am Boden heruntergeladen werden. Dies bedeutet auch, dass jeweils Modelle mit verschiedenen Initialisierungszeitpunkten heruntergeladen wurden. Aus Anonymitätsgründen können die Landezeit und das genaue Vorhersagemodell nicht angegeben werden.

Sämtliche beachteten Flüge waren Anflüge auf den Flughafen Zürich, daher stammen alle Vorhersagedaten vom ICON-D2-Modell. Die Daten wurden nicht interpoliert, durch den Gebrauch des `codes_grib_find_nearest`-Befehls werden die Werte des nächstgelegenen Gitterpunkts benutzt.

Da die Daten teilweise nicht in denselben Einheiten gespeichert waren, mussten einige Parameter umgerechnet werden. Bei den gemessenen Flugdaten wurden die Höhe von Fuss in Meter umgewandelt und die Umgebungstemperatur von Grad Celsius in Grad Kelvin. Reinert et al. (2021) beschreiben in ihrer Dokumentation der Einheiten des ICON, dass der Druck in Pascal angegeben wird und der Wind in u- und v-Richtung in Metern pro Sekunde. Der Druck musste somit in Hektopascal umgewandelt werden. Der Wind musste zuerst in Knoten umgerechnet und danach in Windrichtung und -geschwindigkeit aufgeteilt werden. Die Windrichtung konnte mit folgender Formel ermittelt werden (ECMWF, 2022):

$$\phi = \mod\left(180 + \frac{180}{\pi}\arctan 2(v, u), 360\right)$$

Dabei gibt ϕ die Richtung in Grad zwischen 0 und 360 an, woher der Wind kommt. Die Windgeschwindigkeit wiederum wurde mit dem Satz von Pythagoras $|\vec{V}| = \sqrt{u^2 + v^2}$ ermittelt.

Anschliessend konnten die gemessenen und die heruntergeladenen Daten verglichen werden. Dabei wurden mit der Statistiksoftware R Box-Whiskerplots erstellt. Abbildung 3.3 zeigt eine Beschreibung dieser Plots. Zwischen dem 25 %-Quartil Q1 und dem 75 %-Quartil Q3 liegt die Hälfte der Daten. Der Abstand vom Minimum zu Q1 respektive vom Maximum zu Q3 beträgt anderthalb mal dem IQR. Ausreisser werden ausserhalb dargestellt. Q2 beschreibt den Median. Die Kerbe (Notch) beinhaltet das 95 %-Konfidenzintervall des Medians. Wenn sich also die Kerben zweier Gruppen überlappen, dann bedeutet dies, dass sich die Gruppen nicht signifikant unterscheiden.

Die Resultate der Validierung sind im Abschnitt 4.4 beschrieben.

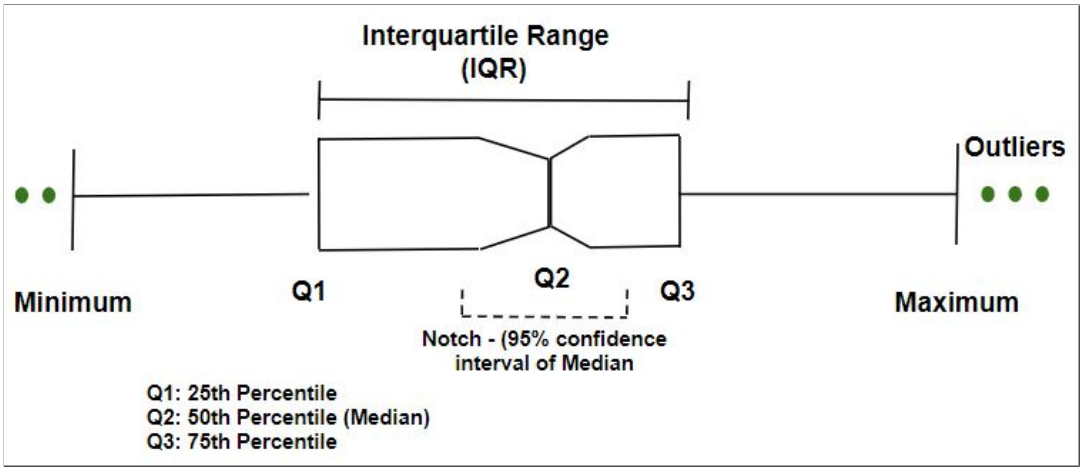


ABBILDUNG 3.3: Beschreibung eines Box-Whiskerplots mit Kerbe (aus Gupta, 2019).

Kapitel 4

Resultate

4.1 Python-Skript

Dieser Abschnitt beschreibt den Workflow des Python-Skripts. Dabei werden die wichtigsten Funktionen näher beschrieben und ihre Arbeitsweise erklärt. Im Anhang A ist zudem ein Flowchart des Skripts abgebildet, das den Ablauf graphisch aufzeigt. Anhang B enthält den gesamten Source Code.

Im Sinne von OOP und aufgrund der variablen Eigenschaften der URL wurde während der Entwicklung darauf geachtet, das Skript möglichst in objektorientierter Form zu schreiben. Somit sind zukünftige Änderungen einfacher zu implementieren und das Skript behält eine übersichtliche Struktur. Daher sind die meisten Aufgaben in einer eigenen Funktion geschrieben, welche dann von der `main()`-Funktion aufgerufen werden.

Initialisierung

Die `main()`-Funktion wird aufgerufen, wenn das Skript mit run gestartet wird. Als erstes wird die Funktion `download_HHL()` abgefragt. Diese zählt mit einer for-Schleife anhand der Dateinamen alle HHL-Dateien, die sich im selben Ordner befinden wie das Skript. Wie im Abschnitt 2.1.4 beschrieben, gibt es für das ICON-D2-Modell 66 HHL-Dateien, für das ICON-EU 61 Dateien. Falls mehr als 126 Dateien vorhanden sind, wird die Funktion mit `return` beendet. Sollten nicht genügend Dateien vorhanden sein, müssen sie heruntergeladen werden. Dafür werden zuerst mit der `round_down_time()`-Funktion (siehe weiter unten) die Bausteine für die URL generiert. Anschliessend werden in einer for-Schleife mit folgendem Befehl die Dateien heruntergeladen und benennt:

```
urllib.request.urlretrieve(url, f"D2_HHL_level_{i}.grib2.bz2")
```

In der gleichen for-Schleife wird jede heruntergeladene Datei mit der Funktion unzip_file() entpackt.

Danach werden in der `main()`-Funktion die benötigten Variablen sowie die Punkte, für welche diese ausgelesen werden sollen, definiert:

```
variables_of_interest = ["t", "p", "qv", "u", "v", "w"]
points_in_space = ((41.3003, 2.0920, 2000),
                   (41.3003, 2.0920, 1000, 2021, 12, 17, 22, 55))
```

for-Schleife über die Punkte

In einer grossen for-Schleife, welche praktisch den gesamten restlichen Code der `main()`-Funktion enthält, wird über jeden definierten Punkt im `points_in_space`-Tupel iteriert. Als Voreinstellung wird zu Beginn die globale Variable `ICON_switcher` auf D2 gesetzt. Danach werden die Punkte in die Variablen `lat, lon` und `alt` aufgeteilt und die Zeitangaben in das *datetime*-Format umgewandelt. Falls keine Zeit angegeben wurde, wird die Zeitvariable auf 0 gesetzt. Weiter werden mit einer doppelten if-Abfrage die Koordinaten überprüft. In Abhängigkeit des Resultats wird der `ICON_switcher` entweder auf D2 belassen oder auf EU geändert. Sollten die angegebenen Koordinaten ausserhalb beider Bereiche liegen, wird eine Fehlermeldung ausgegeben und der Code abgebrochen.

Im nächsten Schritt wird der Index für den Punkt ermittelt. Dabei ruft die `main()`-Funktion die Funktion `get_index_from_gribfile()` auf. In dieser wird die separat gespeicherte `eccodes_get_nearest.py`-Datei geöffnet. Diese Datei enthält den ecCodes-Befehl `codes_grib_find_nearest()` und alle dafür benötigten Abhängigkeiten. Der Befehl gibt den Index jenes Gitterpunktes heraus, der am nächsten bei den angegebenen Koordinaten liegt.

Als nächstes wird die Modellebene ermittelt. Aus der `main()`-Funktion wird dafür die Funktion `get_modellevel_from_altitude()` aufgerufen. Diese benötigt als Argumente den Index und die Höhe. Mit einer while-Schleife werden aus allen HHL-Dateien die Werte am angegebenen Index ausgelesen und in eine Liste gespeichert. Das Auslesen geschieht mit dem ecCodes-Befehl `codes_get_values()`, welcher in der Funktion `read_value_from_gribfile()` gespeichert ist. Die Liste mit den Höhen der Ebenen wird mit der angegebenen Höhe verglichen und die am nächsten liegende Modellebene wird ausgegeben. Mit dem Index und der Modellebene können nun die Variabledateien heruntergeladen und ausgelesen werden.

for-Schleife über die Variablen

Eine kleinere for-Schleife iteriert innerhalb der grösseren Schleife der Punkte über alle Variablen in der `variables_of_interest`-Liste. Als Voreinstellung wird zu Beginn die globale Variable `oldermodel` auf falsch gesetzt. Es wird überprüft, ob die benötigte Variabledatei bereits im selben Ordner wie das Skript vorhanden ist. Dafür wird aus der `main()`-Funktion die Funktion `build_url()` aufgerufen. Innerhalb dieser wird die `round_down_time()`-Funktion verwendet. Diese Funktion benötigt als Argument die aktuelle oder die für einen Punkt angegebene Zeit. Daraus ermittelt sie den aktuellen Initialisierungszeitpunkt der ICON-Modelle durch folgenden Code:

```
if a == 0 or a == 3 or a == 6 or a == 9 or a == 12 or a == 15
          or a == 18 or a == 21:
    rounder = 0
elif a == 1 or a == 4 or a == 7 or a == 10 or a == 13 or a == 16
            or a == 19 or a == 22:
    rounder = 1
else:
    rounder = 2

rounded_time = actual_time.replace(microsecond=0, second=0, minute=0)
                 - timedelta(hours=rounder)
```

Hierbei ist `a` die Stunde der aktuellen Zeit in UTC. Der `rounder` wird anschliessend noch angepasst, falls für den Punkt eine Zeit angegeben wurde. Somit definiert dieser die Vorhersagestunde. `rounded_time` und `rounder` werden in Strings umgewandelt und dann zurückgegeben. Die `build_url()`-Funktion kann mit diesen Angaben sowie den Argumenten Modellebene und Variable den Dateinamen und die URL bauen:

```
filename = f"icon-d2_germany_regular-lat-lon_model-level_{date}
                {hour}_{forecast_time}_{modellevel}_
                {variable}.grib2.bz2"
url = f"https://opendata.dwd.de/weather/nwp/icon-d2/grib/{hour}
                /{variable}/{filename}"
```

Beide Argumente werden in die `main()`-Funktion zurückgegeben. Falls die gesuchte Datei bereits vorhanden ist, wird der Variablewert mit der oben beschriebenen Funktion `read_value_from_gribfile()` aus der Datei gelesen und gedruckt. An diesem Punkt ist das Ende beider for-Schleifen erreicht. Dies bedeutet, dass sie entweder wieder von vorne beginnen, oder aber der Code ist zu Ende.

Falls die gesuchte Datei nicht vorhanden ist, wird die globale Variable `oldermodel` auf richtig gesetzt und weiter nach einer Datei gesucht. Dafür wird die Funktion `build_url()` nochmals gestartet. Durch die globale Variable wird die Zeit in der `round_down_time()`-Funktion um drei Stunden zurückgesetzt. Dies entspricht dem nächstälteren Initialisierungszeitpunkt der ICON-Modelle. Falls nun eine Datei gefunden wird, wird wiederum die Variable ausgelesen und gedruckt.

Sollte noch immer keine Datei gefunden worden sein, wird die globale Variable `oldermodel` wieder auf falsch gesetzt und die `download()`-Funktion aufgerufen. Innerhalb dieser wird mittels `build_url()` die URL gebaut. Anschliessend wird mit dem Befehl `urllib.request.urlretrieve()` die Datei heruntergeladen. Danach wird sie entpackt und ausgelesen.

Am Ende der jeweiligen for-Schleifen werden die ausgelesenen Variablen in Listen gespeichert. Diese Listen werden am Ende gesammelt und danach der Funktion `write_to_csv()` als Argument übergeben. Die Funktion schreibt alle Inhalte der Listen in eine csv-Datei und speichert diese im selben Ordner, wie das Skript gespeichert ist.

Optional kann die Funktion `remove_old_files()` aus der `main()`-Funktion aufgerufen werden. Diese durchsucht den Ordner, in dem das Skript gespeichert ist, nach Variabledateien. Falls eine Datei vor mehr als vier Stunden heruntergeladen wurde, wird sie gelöscht.

4.2 Output

Das Python-Skript gibt die ausgelesenen Daten in zwei verschiedenen Formen an. Die erste Form ist die Ausgabe im „run-tool“-Fenster. In der grösseren for-Schleife der `main()`-Funktion sind zwei `print`-Statements geschrieben, welche die Angaben zum jeweiligen Punkt ausgeben. Zuerst wird die angegebene Zeit gedruckt, gefolgt vom Punkt selbst, dem korrespondierenden Modell und der verwendeten Modellebene. In der kleineren for-Schleife, welche über die Variablen iteriert, ist an deren Ende ein weiteres `print`-Statement geschrieben. Dieses gibt den Variablenamen und deren Wert aus. Für jeden Punkt resultiert folgende Ausgabe:

```
2022-01-20 19:06:00
Point: (47.4578, 8.55115, 400, 2022, 1, 20, 19, 6)
        // Model taken: D2 // Level: 65
t =  274.4001770019531
p =  97635.8125
qv =  0.003850374312605709
u =  0.6827430725097656
v =  -0.9815731048583984
w =  0.005435466766357422
```

Die zweite Form der Ausgabe ist die csv-Datei. Die im Abschnitt 4.1 beschriebene Funktion `write_to_csv()` schreibt einen Header in die erste Zeile der Datei. Dieser beschreibt die Daten der folgenden Zeilen. Anschliessend werden in jeder Zeile die Daten für einen Punkt angegeben. In den ersten fünf Spalten stehen die Angaben zum Punkt selbst. Die weiteren sechs Spalten enthalten die ausgelesenen Werte der Variablen. Als Name dieser Datei wird die aktuelle Zeit angegeben.

4.3 Performance

Dieser Abschnitt beschreibt die Resultate der Performance-Analysen und der daraus resultierenden Verbesserungen. In Tabelle 4.1 sind die Laufzeiten verschiedener Funktionen aufgelistet. Dabei sind in der ersten Spalte die Zeiten aufgeführt, die das Skript in der ersten Fassung benötigte, also bevor die Verbesserungen aus Abschnitt 3.3 implementiert wurden. Die zweite Spalte zeigt die Zeiten, nachdem die erste Verbesserung eingeführt wurde. Dabei wurde überprüft, ob Dateien bereits vorhanden sind, bevor sie heruntergeladen werden. In der dritten Spalte werden jene Zeiten aufgeführt, die nach der zweiten Modifikation entstanden. Bei dieser wurde die Anzahl Benutzungen des ecCodes-Befehls `codes_get_values()` reduziert.

Funktion oder Unterfunktion	Ursprüngliche Version	Erste Verbesserung	Zweite Verbesserung
Ganzes Skript	14 s - 18.8 s	2.9 s - 3.5 s*	0.5 s - 0.6 s*
`download_HHL()`	9.5 s - 12.9 s	–	–
`main()` ohne HHL	4.5 s - 5.5 s	2.9 s - 3.5 s*	0.5 s - 0.6 s*
`get_modellevel()`	2.4 s - 2.8 s	2.4 s - 2.8 s	0.1 s - 0.2 s
get variable	1.3 s - 1.6 s	0.4 s - 0.5 s*	0.3 s*
download file	0.7 s - 0.8 s	–*	–*
unzip file	0.2 s - 0.4 s	–*	–*
`_find_nearest()`	0.3 s - 0.4 s	0.3 s - 0.4 s	–

TABELLE 4.1: Laufzeiten verschiedener Funktionen für einen Punkt. Mit einem Stern (*) gekennzeichnete Felder variieren, jenachdem ob Dateien heruntergeladen werden müssen.

Die Angaben in der Tabelle beziehen sich jeweils auf die Laufzeiten, wenn ein einzelner Punkt ausgelesen wird. Die Verbesserungen zeigen den bestmöglichen

Fall, wenn alle Dateien bereits vorhanden sind. Ist dies nicht der Fall, müssen die Dateien heruntergeladen und entpackt werden. Dies bedeutet, dass die Zeiten sich entsprechend verlängern. Die betroffenen Felder sind in der Tabelle deshalb mit einem Stern (*) markiert. Für die HHL-Dateien gilt dies im Allgemeinen nicht, da diese nur bei der allerersten Ausführung des Skripts in einem neuen Speicherort heruntergeladen werden. Für die Performance-Analyse kann deshalb davon ausgegangen werden, dass sie bereits vorhanden sind.

Die zweite Verbesserung war der Ersatz des Befehls `codes_grib_find_nearest` durch den Befehl `codes_get_values`. Die Laufzeit des Letzteren ist praktisch vernachlässigbar. Dies erklärt den deutlichen Fortschritt beim Auslesen der Modellebene sowie den fehlenden letzten Eintrag in der Tabelle (siehe dazu auch Abbildung 3.2 im vorherigen Kapitel).

4.4 Validierung

Im Abschnitt 3.4 wurde beschrieben, mit welchen Methoden die von Flugzeugen gemessen Daten mit den vom Skript ausgegebenen Daten verglichen wurden. Die daraus entstandenen Validierungsdatensätze werden hier präsentiert. Um sie graphisch darzustellen, wurden Box-Whiskerplots mit Kerbe erstellt. Eine Auswahl dieser Graphiken wird im folgenden Abschnitt genauer analysiert. Abgebildet sind jeweils der vom Flugzeug gemessene Datensatz in blau sowie zwei Modellvorhersagen in orange und rot.

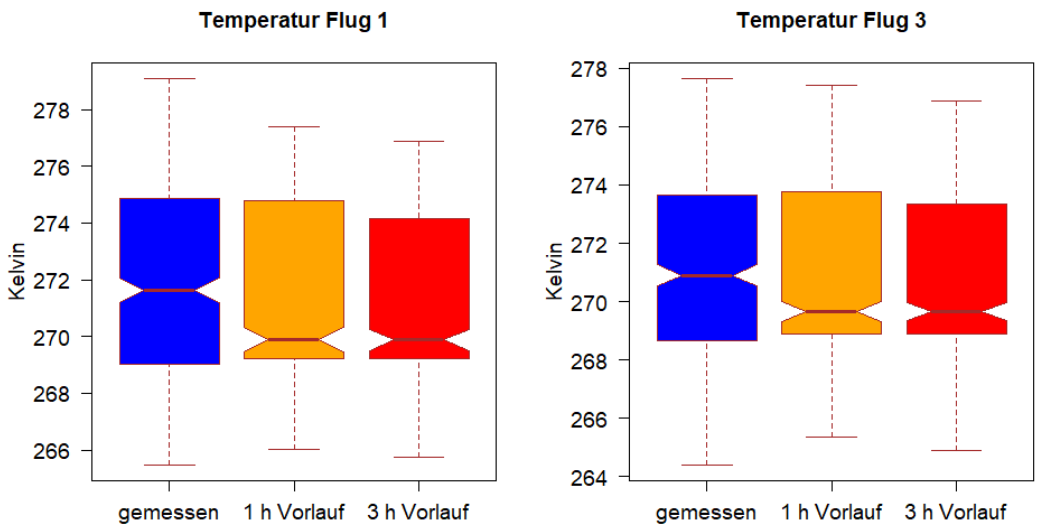


ABBILDUNG 4.1: Gemessene und vorhergesagte Temperatur für Flug 1 und Flug 3.

Abbildung 4.1 zeigt im linken Teil einen Box-Whiskerplot für die Temperaturmessungen des ersten Fluges. Die erste Vorhersage (orange) wurde eine Stunde vor dem Zeitpunkt der Landung heruntergeladen, die Zweite (rot) drei Stunden vorher. Bei diesem Beispiel wurden 421 Datenpunkte miteinander verglichen. Die Einheit der Daten ist Kelvin. Bei den gemessenen Daten (blau) liegt der Median bei 271.62,

das erste Quartil Q1 bei 269.03 und das dritte Quartil Q3 bei 274.88. Die erste Vorhersage hat einen Median von 269.88, die Quartile liegen bei 269.24 respektive bei 274.77. Die zweite Vorhersage hat den selben Median und das selbe Q1 wie die erste, Q3 liegt bei 274.17.

Rechts in Abbildung 4.1 ist ein weiteres Beispiel für Temperaturdifferenzen zu sehen, diesmal für den dritten Flug. Die Messreihen beinhalten jeweils 504 Punkte. Die gemessenen Daten weisen einen Median von 270.9 auf, die Hälfte der Daten liegt zwischen 268.65 und 273.65. Die Vorhersagedaten weisen beide den selben Median von 269.66 sowie das selbe Q1 von 268.89 auf. Das Q3 liegt bei 273.78 respektive bei 273.35.

Für den Luftdruck wird ein Beispiel von Flug 4 in Abbildung 4.2 gezeigt. Die Vorhersagen wurden zwei Stunden (orange) respektive vier Stunden (rot) vor dem Zeitpunkt der Landung heruntergeladen. Sie beinhalten jeweils 478 Datenpunkte, welche in Hektopascal angegeben werden. Der Median der gemessenen Daten (blau) liegt bei 865.43, Q1 und Q3 liegen bei 789.47 respektive bei 946.74. Die beiden Vorhersagen weisen einen Median von 863.89 respektive 864.20 auf. Deren erste Quartile liegen bei 788.76 respektive 788.83, die dritten Quartile bei 944.71 sowie bei 945.29.

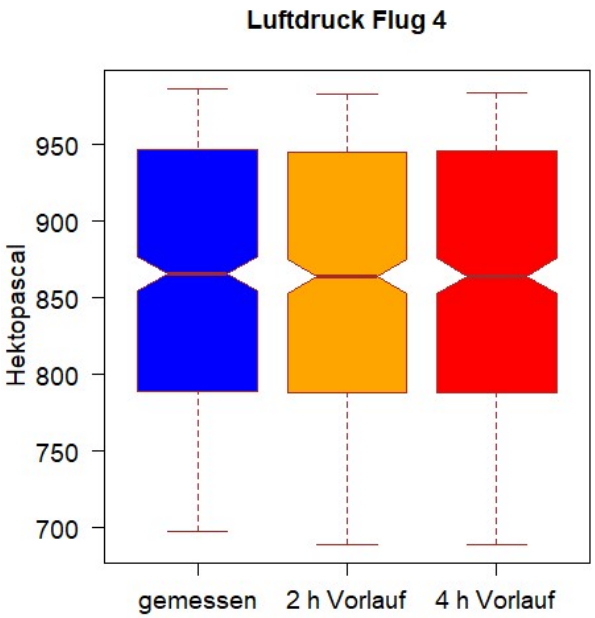


ABBILDUNG 4.2: Gemessener und vorhergesagter Luftdruck für Flug 4.

Abbildung 4.3 zeigt links die Unterschiede der Windgeschwindigkeiten zwischen den gemessenen Daten und den Vorhersagen für den fünften Flug. Die heruntergeladenen Modelldaten stammen von zwei Stunden (orange) respektive fünf Stunden (rot) vor der Landung und beinhalten 496 Datenpunkte. Die Windgeschwindigkeit wird in Knoten angegeben. Die gemessenen Daten (blau) zeigen einen Median von 8, die Hälfte der Daten liegt zwischen 5 und 12. Bei der Vorhersage, die zwei Stunden vorher heruntergeladen wurde, liegt der Median bei 6.1, Q1 und Q3 liegen bei 4.54 respektive bei 9.81. Die ältere Vorhersage weist einen Median von 8.75 auf, mit den 25 und 75 %-Quartilen bei 5.51 sowie bei 10.96.

Die Daten der Windrichtung von Flug 1 sind in Abbildung 4.3 rechts gezeigt. Die jeweils 421 Datenpunkte der Vorhersagen wurden – analog zum Beispiel der Temperatur – eine Stunde (orange) respektive drei Stunden (rot) vor der Landung heruntergeladen. Die Windrichtungen sind in Grad angegeben. Bei den gemessenen Daten (blau) liegt der Median bei 315.44, die Hälfte der Daten liegt zwischen 299.16 und 323.92. Die Vorhersagen weisen den selben Median und das selbe Q3 auf; 328.72

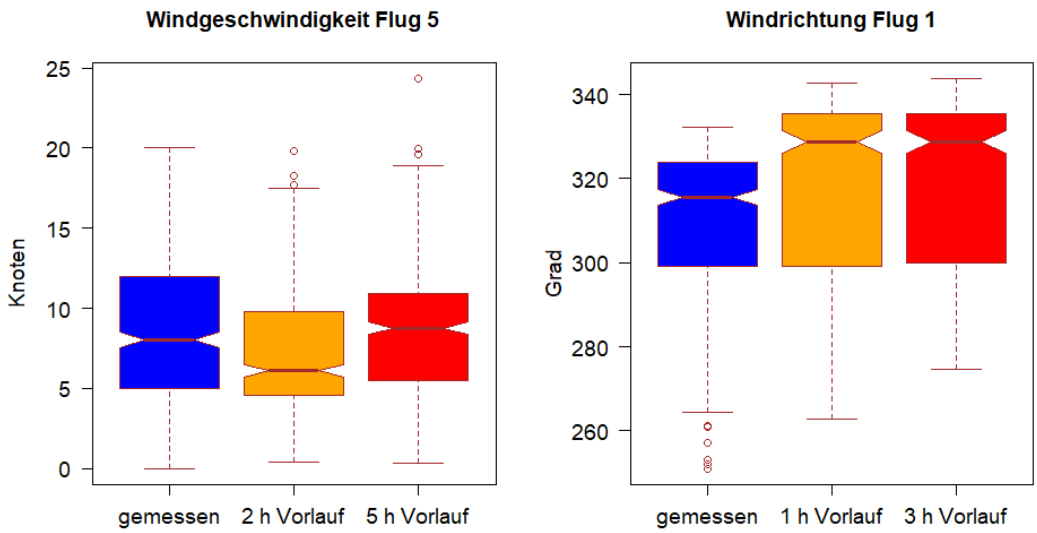


ABBILDUNG 4.3: Gemessene und vorhergesagte Daten für Flüge 5 und 1.

respektive 335.47. Das Q1 liegt bei der ersten Vorhersage bei 299.16, bei der zweiten liegt es bei 299.90.

4.5 Spinoff ICON-Downloader

Auf Anfrage des Betreuers hat der Autor ein weiteres Pythonskript entwickelt, welches auf dem Hauptskript basiert. Die Anforderung war, für das Zentrum für Aviatik an der ZHAW einen ICON-Downloader zu bauen, welcher alle auf dem Open-Data-Server des DWD (2020) verfügbaren Dateien des ICON-EU- sowie des ICON-D2-Modells herunterlädt. Dafür konnten einige Bausteine des Hauptskripts leicht angepasst wiederverwendet werden. Im Folgenden wird kurz der Aufbau beschrieben:

Die `main()`-Funktion enthält fünf ineinander verschachtelte for-Schleifen. In diesen wird jeweils über die beiden Modelle, die sechs Variablen, die sieben Initialisierungszeitpunkte, die 48 Vorhersagestunden und alle jeweiligen Modellebenen iteriert. Das Ausführen dieser Serie von Schleifen resultiert in rund 250'000 herunterzuladenden Dateien. Bevor sie heruntergeladen werden, wird wie beim Hauptskript überprüft, ob sie bereits offline vorhanden sind. Am Ende zeigt das Skript an, wie viele Dateien tatsächlich heruntergeladen wurden.

Die aus der `main()`-Funktion aufgerufene Funktion `create_timelist()` basiert auf der `round_down_time()`-Funktion. Sie erstellt eine Liste aller Initialisierungszeitpunkte in den letzten 21 Stunden. Die `build_url()`-Funktion sowie die Entpackungsfunktion arbeiten gleich wie im Hauptskript. Ebenfalls hinzugefügt wurde die `remove_old_files()`-Funktion, wobei diese mit einer zusätzlichen Warnung versehen wurde. Die Warnung verhindert, dass versehentlich alle Dateien gelöscht werden. Sie muss manuell angepasst werden, bevor sich die Dateien überhaupt entfernen lassen.

Die Vorhersagedaten auf dem Open-Data-Server werden alle 24 Stunden überschrieben. Um also zu garantieren, dass alle Dateien heruntergeladen werden, sollte das Skript mindestens zwei Mal täglich ausgeführt werden. Der ICON-Downloader wurde am 29. 12. 2021 via GitHub dem Betreuer übergeben. Inkludiert war dabei auch eine README-Datei.

Kapitel 5

Diskussion

Diese Arbeit hatte gemäss Aufgabenstellung zwei grundlegende Ziele. Das erste war, eine Literaturrecherche zu den Themen deterministische und probabilistische Wettervorhersage sowie zum LNAS-Projekt durchzuführen. Dabei sollten ein Überblick verschafft und die zentralen Aspekte zusammengefasst werden. Diese Aspekte sind in Kapitel 2 aufgeführt. Aus Sicht des Autors sind die wesentlichen Charakteristiken gefunden und diskutiert worden.

Das zweite Ziel war, eine Schnittstelle zu entwickeln, welche basierend auf einer Flugplandatei die entsprechenden 3D-Wetterdaten von Vorhersageprodukten wie dem ICON-Modell herunterlädt und einem Anwender zur Verfügung stellt. Dafür wurde ein Pythonskript programmiert, welches diese Aufgaben weitgehend erfüllt. In den folgenden Abschnitten werden die verschiedenen Arbeitsschritte kritisch betrachtet.

Skript

Das Pythonskript ist das eigentliche Resultat dieser Arbeit. Es ermittelt anhand der Eingabe der Punkte, welche Vorhersagedateien benötigt werden. Anschliessend holt es sich diese Dateien, entweder vom lokalen Speicherort, oder es lädt diese vom Open-Data-Server des DWD herunter und entpackt sie. Das Auslesen der Werte aus den grib2-Dateien erfolgt mit den ecCodes-Befehlen `codes_grib_find_nearest` und `codes_get_values`. Die Ausgabe der ausgelesenen Werte folgt mit mehreren `print`-Statements sowie einer Speicherung in eine csv-Datei.

Der Autor vertritt die Meinung, dass das Skript mit diesen Funktionen zumindest den Basisanforderungen gerecht wird. Für die Implementierung in das LNAS-Projekt ist gemäss dem Zweitbetreuer das DLR-Projektteam verantwortlich. Dies bedeutet, dass keine grossen Anforderungen an Form der Ein- und Ausgabe gestellt waren.

Nichtsdestotrotz ist gerade bei der Eingabe der Punkte in das Skript ein gewisses Verbesserungspotenzial vorhanden. Aktuell müssen für jeden Punkt einzeln jeweils manuell die Koordinaten, die Höhe und – falls benötigt – die Zeit in das Skript eingegeben werden. Hier wäre beispielsweise eine sogenannte Waypointsfunktion wünschenswert. Bei dieser wären die Koordinaten aller Punkte von verschiedenen Arrivalprocedures und Transitions für einen Flugplatz gespeichert. Damit müsste als Input nur der Name des Procedures angegeben werden, und das Skript gibt dann die Werte für den ganzen Anflug an.

Eine weitere mögliche Form der Eingabe wäre das Einlesen einer Datei, in der Koordinatenpunkte gespeichert sind. Eine solche Funktion wurde zwar für das Einlesen der Validierungsdateien, welche von SWISS zur Verfügung gestellt wurden,

bereits benutzt. Allerdings ist die Architektur dieser Funktion sehr spezifisch auf diese Dateien zugeschnitten. Sie wäre also nicht für beliebige Dateiformate und Speicherformen anwendbar. Aus diesem Grund wurde davon abgesehen, die Funktion im Skript zu belassen.

Ein weiterer Verbesserungspunkt betrifft die verwendeten Daten. Zurzeit werden lediglich die beiden Vorhersagemodelle ICON-D2 und ICON-EU des DWD benutzt. Eine oder mehrere zusätzliche Datenquellen wären aus zahlreichen Gründen wünschenswert. Kiktev et al. (2017) konnten beispielsweise zeigen, dass die Kombination mehrere Modelle bessere Vorhersagen ergibt. Die Integrität der ausgegebenen Daten könnte damit verbessert werden. Ausserdem bestünde keine Abhängigkeit mehr von einem einzelnen Anbieter. Weiter kann damit auch, je nach Modell, das geographische Einsatzgebiet ausgedehnt werden. Dieses ist im Moment mit dem ICON-EU-Modell auf Europa beschränkt.

Logikfehler im Ablauf

Beim Verfassen der Beschreibung des Pythonskripts ist dem Autor aufgefallen, dass das Skript einen Logikfehler im Programmierablauf enthält. In der kleineren for-Schleife, welche über die Variablen iteriert, wird vor dem Herunterladen einer benötigten Datei überprüft, ob diese bereits offline vorhanden ist. Dabei wird nach der Datei des Modells mit dem aktuellsten Initialisierungszeitpunkt gesucht. Sollte diese nicht vorhanden sein, wird nach einer Datei des Modells mit dem nächstälteren Initialisierungszeitpunkts gesucht. Erst wenn diese ebenfalls nicht vorhanden ist, wird eine neue Datei heruntergeladen. Diese Logik kann unter gewissen Umständen dazu führen, dass eine ältere Datei verwendet wird, obwohl bereits eine neuere auf dem DWD-Server verfügbar wäre.

Dies wäre beispielsweise in folgender fiktiven Situation der Fall: Um 10 Uhr wird die Vorhersage für einen Flug mit Landung um 14 Uhr heruntergeladen. Dies sind Dateien des Modells mit Initialisierungszeitpunkt 9 Uhr und Vorhersagestunde 5. Wenn nun um 13 Uhr nach der Vorhersage für einen anderen Flug gesucht wird, der ebenfalls um 14 Uhr landen soll, errechnet das Skript den aktuellsten Initialisierungszeitpunkt als 12 Uhr und die Vorhersagestunde 2. Falls diese Dateien noch nicht heruntergeladen wurden, wird auf die bereits vorhandenen Dateien des ersten Fluges zurückgegriffen, obwohl die aktuelleren möglicherweise bereits online wären.

Dieser Fehler führt dazu, dass das Skript nicht die optimalen Ergebnisse liefert. Der Autor betrachtet diesen Umstand aber aus folgenden Gründen als akzeptabel: Das oben beschriebene Beispiel zeigt, dass dieser Fall nur selten auftreten wird. Häufig werden für Flüge mit ähnlichen Landezeiten auch die Vorhersagen ungefähr zur selben Zeit heruntergeladen.

Sollte der Fall trotzdem auftreten, dann wäre das Ergebnis aber nicht gross abweichend, da gemäss Skript nur nach einer Datei mit *einem* Initialisierungszeitpunkt früher gesucht wird. Die Datei ist also maximal um drei Stunden veraltet. Die Validierung, beschrieben in Abschnitt 4.4, hat zudem gezeigt, dass der Unterschied zwischen zwei Modellläufen im Allgemeinen nicht sehr gross ist. Auch aus einer leicht veralteten Vorhersage lassen sich immer noch ähnliche Werte herauslesen.

Ausserdem hat dieser Fehler auch einen positiven Einfluss auf die Performance des Skripts: Werden ältere Dateien verwendet, müssen weniger Dateien heruntergeladen werden. Dies hilft wiederum, die Laufzeit des Skripts zu verkürzen.

Performance

Die in Abschnitt 4.3 beschriebenen Resultate der Performance-Analysen zeigen die Verbesserungen der Laufzeit, welche mit den verschiedenen Modifikationen erreicht wurden. Tabelle 4.1 zeigt die Zeiten dazu. Beide Verbesserungsschritte erwirken einen signifikanten Fortschritt.

Der Ersatz des Befehls `codes_grib_find_nearest` durch `codes_get_values` verkürzt die Laufzeit um rund zwei Sekunden pro Datenpunkt. Der Grund, wieso der erste Befehl so viel langsamer ist als der zweite, ist der, dass die `_nearest`-Funktion jeweils die Werte anhand der Koordinaten suchen muss. Wird wie beim zweiten Befehl ein Index angegeben, weiss die Funktion sofort, an welcher Stelle sich der gesuchte Wert befindet.

Der andere Verbesserungsschritt reduziert die Anzahl der Dateien, die heruntergeladen werden, unter Umständen enorm. Dadurch reduziert sich die Laufzeit um über eine Sekunde pro Punkt. Unter optimalen Voraussetzungen dauert die Ausführung des Skripts also 0.5 bis 0.6 Sekunden pro Punkt. Im alltäglichen Betrieb wäre das der Fall, wenn für mehrere Flüge mit ähnlichen geplanten Landezeiten praktisch gleichzeitig die Daten heruntergeladen werden. In einem Wellensystem, welches die SWISS zurzeit auf der Kurzstrecke betreibt, sind diese Voraussetzungen durchaus gegeben.

Für den Fall, dass noch keine Dateien offline verfügbar sind, verlängert sich die Laufzeit natürlich. Doch auch da hält sich die Verlängerung in Grenzen. Grund dafür ist, dass die Dateien der ICON-Modelle in Modellebenen gespeichert sind. Zwischen dem Boden und einer Höhe von FL100 befinden sich je nach Gelände rund 30 Ebenen. Wenn also für einen Flug beispielsweise 200 Punkte angegeben werden, verlängert sich die Laufzeit nicht um 200 Sekunden, sondern nur um 30 bis 45 Sekunden.

Eine zusätzliche Idee, wie die Performance des Skripts noch verbessert werden könnte, ist die Parallelität. Es könnte nach Möglichkeiten gesucht werden, um das Herunterladen und das Auslesen gleichzeitig vorzunehmen. Ausserdem gibt es für Python gewisse Befehle, die einen parallelen Download mehrerer Dateien erlauben (Gorelick & Ozsvald, 2020).

Validierung

Das in dieser Arbeit entwickelte Skript lädt öffentlich verfügbare Wettervorhersagedaten herunter und zeigt diese in geeigneter Form an. Im Prinzip ist es also eigentlich nur ein Anwendetool, das bereits vielfach überprüfte und getestete Daten anzeigt. Der DWD (2021c) betreibt das erste ICON-Modell bereits seit 2015 und entwickelt dieses seither ständig weiter. Aus diesem Grund wurde zu Beginn dieser Arbeit zu wenig Wert auf eine Validierung der Daten gelegt.

Um aber trotzdem zu zeigen, dass die angezeigten Daten zumindest realistisch und für verschiedene Flugwege anwendbar sind, wurden einige Messdaten von Flugzeugen als Vergleich verarbeitet. Die dabei entstandenen Resultate wurden im Abschnitt 4.4 beschrieben.

Die Box-Whisker-Plots zeigen, dass die Modelle die Realität teils relativ genau vorhersagen können. Insbesondere bei den Vorhersagen des Luftdrucks sind die Abweichungen sehr gering. Dabei zeigt sich in Abbildung 4.2 auch, dass sich die

jeweiligen Kerben in den Boxplots grösstenteils überschneiden. Dies bedeutet, dass die Vorhersagen für den Luftdruck nicht signifikant von den Messungen abweichen, sie also die Realität sehr genau modellieren können.

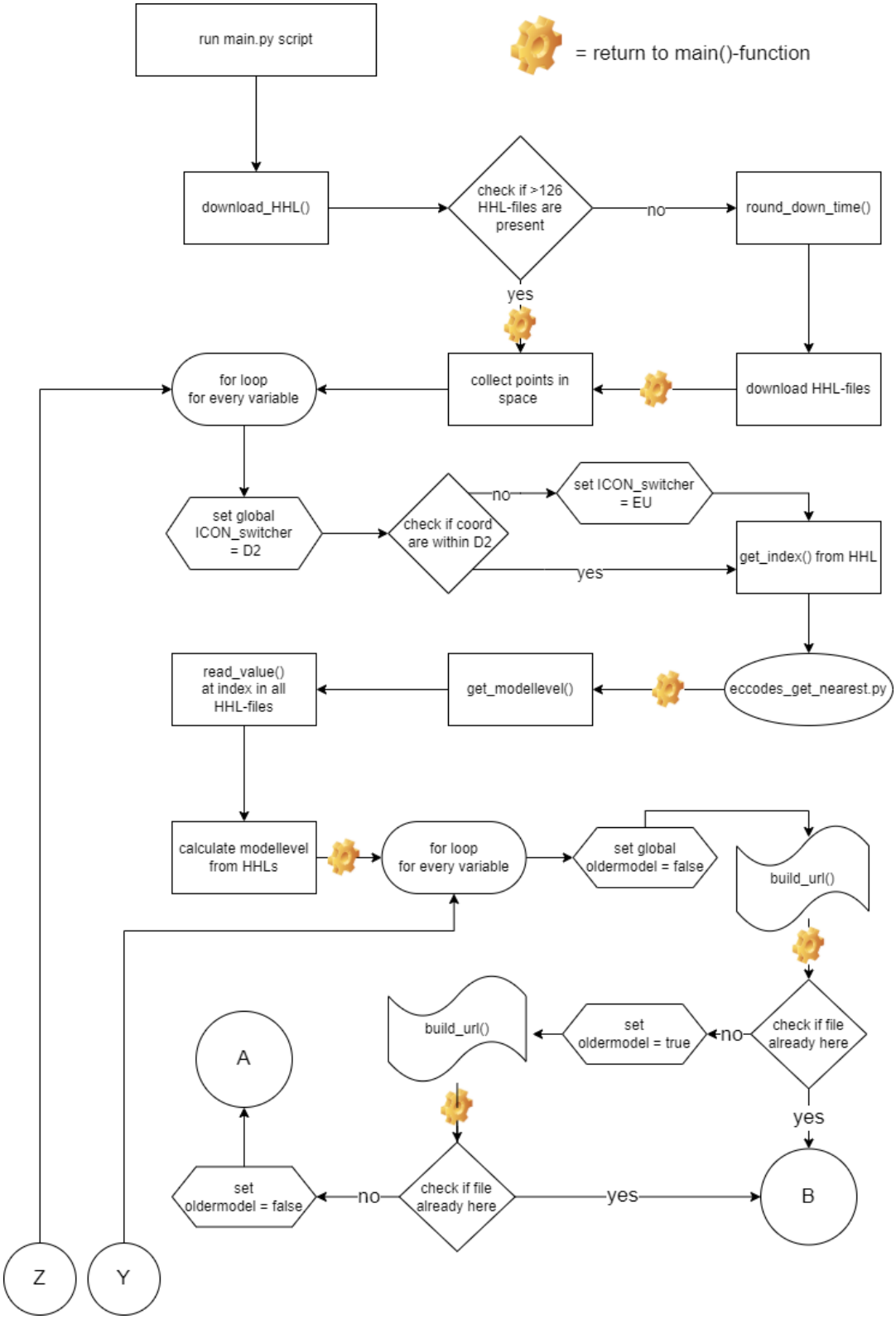
Bei den Parametern Temperatur, Windgeschwindigkeit und Windrichtung weichen die Vorhersagen etwas mehr von den Messungen ab. Auch wenn sich die Bereiche zwischen dem ersten und dem dritten Quartil oft grossflächig überschneiden, gilt dies überaschenderweise nicht für den Median sowie die Kerbe. Die Daten der Vorhersagen unterscheiden sich also signifikant von den Messdaten.

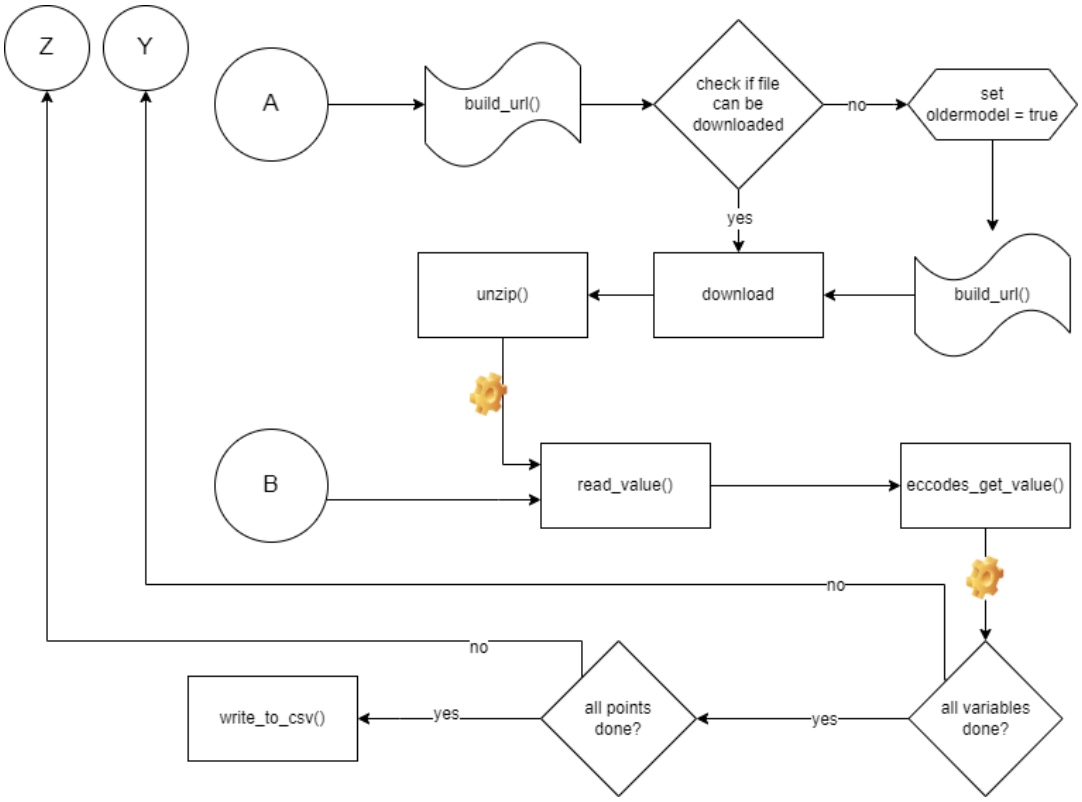
Die Frage, weshalb sich die Daten tatsächlich in diesem Ausmass unterscheiden, kann nicht abschliessend beantwortet werden. Eine Bestätigung der oben gemachten Befunde mittels gepaartem T-Test war nicht zielführend, da die Daten nicht Normalverteilt sind. Auch ein Versuch mit dem Wilcoxontest konnte keine weiteren Erkenntnisse liefern. Der Grund dafür könnte darin liegen, dass die Menge an verfügbaren Daten zu klein war.

Noch gar nicht untersucht wurde die – aus Sicht des Autors – viel wichtigere Fragestellung, ob die Vorhersagedaten auch einen Nutzen für das LNAS-System bereiten. Oder umgekehrt, ob das LNAS eine bessere Leistung erbringen würde, wenn die Wetterdaten mit einbezogen werden. Dies bedingt eine weiterführende Validierung der Daten, vorzugsweise mit einer Implementierung ins LNAS. Ausserdem wurde der Unterschied zwischen dem ICON-D2 und dem ICON-EU Modell noch nicht analysiert. Diese extensiven Validierungen sollen Bestandteil der anschliessenden Vertiefungsarbeit 2 im dritten Semester sein.

Anhang A

Flowchart




Z
Y
A
build_url()
check if file can be downloaded
no
set oldermodel = true
yes
unzip()
download
build_url()
B
read_value()
eccodes_get_value()
no
no
write_to_csv()
yes
all points done?
yes
all variables done?

Anhang B

Source Code

```python


import urllib.request
from datetime import datetime, timedelta
import time
import bz2
import os
import sys
import eccodes_get_nearest
from eccodes import *
import fnmatch
import csv


##################################
#          Download HHL          #
##################################

def download_HHL():
    """
    Checks if enough HHL-files are present. If not it downloads them.
    """

    HHL_counter = 0
    for file in os.listdir('.'):
        if fnmatch.fnmatch(file, '*HHL_level*'):
            HHL_counter += 1

    if HHL_counter > 126:
        return

    global oldermodel                   # HHL is time-invariant, so it's not necessary to check whether the latest model
    oldermodel = True                   # is available. It's easier to just download an older file.

    rounded_time = round_down_time(0)

    date = rounded_time[0][0:8]
    hour = rounded_time[0][9:11]

    for i in range(1, 67):
        url = f"https://opendata.dwd.de/weather/nwp/icon-d2/grib/{hour}/hhl/icon-d2_germany_regular-lat-lon" \
              f"_time-invariant_{date}{hour}_000_{i}_hhl.grib2.bz2"

        urllib.request.urlretrieve(url, f"D2_HHL_level_{i}.grib2.bz2")

        unzip_file(f"D2_HHL_level_{i}.grib2.bz2")
```

```
        i += 1

    for i in range(1, 62):
        url = f"https://opendata.dwd.de/weather/nwp/icon-eu/grib/{hour}/hhl/icon-eu_europe_regular-lat-lon_" \
              f"time-invariant_{date}{hour}_{i}_HHL.grib2.bz2"

        urllib.request.urlretrieve(url, f"EU_HHL_level_{i}.grib2.bz2")

        unzip_file(f"EU_HHL_level_{i}.grib2.bz2")
        i += 1


##################################
#         Getting index          #
##################################

def get_index_from_gribfile(file, lat, lon):
    """
    Calls eccodes_get_nearest.py which gives the index of the nearest point for any lat-lon-coordinate.
    """
    index = eccodes_get_nearest.main(file, lat, lon)
    return index


##################################
#         Get Modellevel         #
##################################

def get_modellevel_from_altitude(index, alt):
    """
    Returns the fulllevel which is closest to the given altitude at given index. This function works with a try/except
    block, because the ICON-D2 and the ICON-EU models do not have the same amount of levels.

    :param index
    :param alt
    :return: level
    """

    HHLs = []                                       # List of altitudes of all halflevels
    HFLs = []                                       # List of altitudes of all fulllevels

    i = 1
    while True:                                     # Get values for all halflevels
        try:
            HHL = read_value_from_gribfile(f"{ICON_switcher}_HHL_level_{i}.grib2", index)
            HHLs.append(HHL)
        except:
            break
        finally:
            i += 1

    for i in range(0, len(HHLs) - 1):                           # Calculate fulllevels from halflevels
        HFL = (HHLs[i] + HHLs[i+1])/2
        HFLs.append(HFL)
```

```
    level = min(range(len(HFLs)), key=lambda i: abs(HFLs[i] - alt)) + 1

    return level


##################################
#           Download             #
##################################

def download(lvl, var, time_at_point):
    """
    Checks if latest model is already available and downloads it. If not, it downloads the previous model.
    """

    try:
        url, filename = build_url(lvl, var, time_at_point)
        urllib.request.urlretrieve(url, filename)
    except:
        global oldermodel
        oldermodel = True

        url, filename = build_url(lvl, var, time_at_point)
        urllib.request.urlretrieve(url, filename)

    unzip_file(filename)


def build_url(lvl, var, time_at_point):
    rounded_time = round_down_time(time_at_point)

    date = rounded_time[0][0:8]
    hour = rounded_time[0][9:11]
    forecast_time = str(rounded_time[1]).zfill(3)                # fill with 0 until string is 3 digits long

    modellevel = lvl
    variable = var

    if ICON_switcher == "D2":
        filename = f"icon-d2_germany_regular-lat-lon_model-level_{date}{hour}_{forecast_time}_{modellevel}_" \
                   f"{variable}.grib2.bz2"
        url = f"https://opendata.dwd.de/weather/nwp/icon-d2/grib/{hour}/{variable}/{filename}"
    elif ICON_switcher == "EU":
        filename = f"icon-eu_europe_regular-lat-lon_model-level_{date}{hour}_{forecast_time}_{modellevel}_" \
                   f"{variable.upper()}.grib2.bz2"
        url = f"https://opendata.dwd.de/weather/nwp/icon-eu/grib/{hour}/{variable}/{filename}"

    return url, filename


def round_down_time(time_at_point):
    """
    Takes current UTC and rounds down to a 3hour interval. This is the update cycle of ICON-D2.

    :param time_at_point
    :return rounded_time
```

```
    :return rounder (The amount of hours by which time is rounded down. This is used to get the forecasthour)
    """

    actual_time = datetime.utcnow()
    a = actual_time.hour

    if a == 0 or a == 3 or a == 6 or a == 9 or a == 12 or a == 15 or a == 18 or a == 21:
        rounder = 0
    elif a == 1 or a == 4 or a == 7 or a == 10 or a == 13 or a == 16 or a == 19 or a == 22:
        rounder = 1
    else:
        rounder = 2

    if oldermodel:                                  # if latest model is not yet available take the one from 3 hours before
        rounder += 3

    rounded_time = actual_time.replace(microsecond=0, second=0, minute=0) - timedelta(hours=rounder)

    if type(time_at_point) is datetime:                # if a time is given, take forecast from nearest hour to that time
        difference = time_at_point - rounded_time
        difference = difference.total_seconds()//60
        rounder = str(round(difference / 60))
        if int(rounder) > 24 or int(rounder) < 0:
            sys.exit("Time given is out of window")

    elif actual_time.minute > 30:                        # if no time is given, just take forecast from nearest full hour
        rounder += 1

    rounded_time = str(rounded_time)
    rounded_time = rounded_time.replace("-", "")
    rounded_time = rounded_time.replace(":", "")

    return rounded_time, rounder


##################################
#           Unzipping            #
##################################

def unzip_file(file):
    """
    Takes a bz2-file and unzips it
    """

    filepath = os.path.join(sys.path[0], file)

    comp_file = bz2.BZ2File(filepath)

    data = comp_file.read()                          # get the decompressed data
    newfilepath = filepath[:-4]                      # assuming the filepath ends with .bz2

    open(newfilepath, 'wb').write(data)              # write an uncompressed file

```

```
    os.remove(filepath)                                # remove the compressed file directly


##################################
#            Reading             #
##################################

def read_value_from_gribfile(file, index):
    """
    Returns the value at the given index in given file using ecCodes.

    :param file
    :param index
    :return: value at index
    """

    f = open(os.path.join(sys.path[0], file), 'rb')
    gid = codes_grib_new_from_file(f)

    value = codes_get_values(gid)

    codes_release(gid)
    f.close()

    return value[index]

##################################
#       Write to a csv-file      #
##################################

def write_to_csv(data):
    header = ["Latitude", "Longitude", "geometric Altitude (m)", "UTC", "Level", "t", "p", "qv", "u", "v", "w"]

    time = datetime.utcnow()
    time = str(time.replace(microsecond=0))
    time = time.replace("-", "_")
    time = time.replace(":", "_")
    time = time.replace(" ","_")
    filename = f"{time}.csv"

    with open(filename, 'w', encoding='UTF8', newline='') as f:
        writer = csv.writer(f, delimiter=",")

        # write the header
        writer.writerow(header)

        # write multiple rows
        writer.writerows(data)


##################################
#       Remove old files         #
##################################

def remove_old_files():
    """
    Removes any variable-file that is older than 4 hours
    """

    for file in os.listdir('.'):
        if fnmatch.fnmatch(file, '*regular-lat-lon*'):
```

```
            file_age = (time.time() - os.path.getmtime(file))/3600
            if file_age > 4:
                os.remove(file)


#################################
#              Main             #
#################################

def main():

    download_HHL()

    variables_of_interest = ["t", "p", "qv", "u", "v", "w"]

    """
    Insert here the points of interest, format: latitude, longitude, altitude in meters abv sealevel.
    Optional argument: time within the next 24h in UTC, format  YYYY, MM, DD, HH MM.
    """
    points_in_space = ((41.3003182174652, 2.0920081483251014, 2000), (41.3003182174652, 2.0920081483251014, 1000, 2021, 12, 17, 22, 55))
    # points_in_space = points_simulator()

    csvdata = []

    for point in points_in_space:

        global ICON_switcher
        ICON_switcher = "D2"

        # Decode point-tuples into their parameters:
        lat, lon, alt = point[0], point[1], point[2]
        if len(point) > 3:
            time_at_point = datetime(point[3], point[4], point[5], point[6], point[7])
            print(time_at_point)
        else:
            time_at_point = 0
            print("Current time taken")

        # Check if coordinates are within ICON-D2 range. Otherwise use ICON-EU
        if 44 <= lat <= 50:
            if not 0 <= lon <= 17:
                ICON_switcher = "EU"
        elif 50 < lat <= 57:
            if not -1.5 <= lon <= 18.5 and not 358.5 <= lon:
                ICON_switcher = "EU"
        else:
            ICON_switcher = "EU"

        index = get_index_from_gribfile(f"{ICON_switcher}_HHL_level_1.grib2", lat, lon)

        lvl = get_modellevel_from_altitude(index, alt)

        print("Point:", point, "// Model taken:", ICON_switcher, "// Level:", lvl)

        csvrow = [lat, lon, alt, time_at_point, lvl]

```

```
        for var in variables_of_interest:

            global oldermodel                        # used if latest model is not yet available
            oldermodel = False

            try:                                     # check if file is already present
                filename = build_url(lvl, var, time_at_point)[1][:-4]
                value = read_value_from_gribfile(filename, index)

            except:                                  # if not, check if file of older model is present
                try:
                    oldermodel = True

                    filename = build_url(lvl, var, time_at_point)[1][:-4]
                    value = read_value_from_gribfile(filename, index)

                except:                              # if both files are not yet present, download
                    oldermodel = False

                    download(lvl, var, time_at_point)
                    filename = build_url(lvl, var, time_at_point)[1][:-4]
                    value = read_value_from_gribfile(filename, index)

            print(f"{var} = ", value)

            csvrow.append(value)

        csvdata.append(csvrow)

    write_to_csv(csvdata)


def points_simulator():

    points_in_space = []

    alt = 400
    for i in range(3):
        points_in_space.append((47.45782369382128, 8.551156219956459, alt, 2022, 1, 20, 19, 6))
        alt += 15
        i += 1

    return points_in_space


if __name__ == '__main__':
    sys.exit(main())
```

Abbildungsverzeichnis

2.1 Verbesserung von Vorhersagen seit 1981 3
2.2 Arakawa-C-Gitter (aus ResearchGate, 2021). 5
2.3 Gittermodell mit ikosaedraler Struktur (aus Oeser, 2021). 5

3.1 Beispiel eines Links für eine ICON-D2 Datei. 13
3.2 Ausschnitt einer Analyse mit dem PyCharm-Profiler. 14
3.3 Beschreibung eines Box-Whiskerplots mit Kerbe (aus Gupta, 2019). . . 16

4.1 Gemessene und vorhergesagte Temperatur für Flug 1 und Flug 3. . . . 21
4.2 Gemessener und vorhergesagter Luftdruck für Flug 4. 22
4.3 Gemessene und vorhergesagte Daten für Flüge 5 und 1. 23

Literatur

Abdelmoula, F. & Scholz, M. (2018). LNAS - A PILOT ASSISTANCE SYSTEM FOR LOW-NOISE APPROACHES WITH MINIMAL FUEL CONSUMPTION. *In 31st Congress of the International Council of the Aeronautical Sciences, Belo Horizonte, Brasil (pp. 9-14).*

Abdelmoula, F., Scholz, M. & Kühne, C. G. (2017). *Erprobung und Bewertung des Low Noise Augmentation System (LNAS)* (Techn. Ber.). DLR-IB-FT-BS-2017-15. https://elib.dlr.de/111517/

Abellan, X. (2021). *Python 3 interface for ecCodes* (ECMWF, Hrsg.). Verfügbar 30. Dezember 2021 unter https://confluence.ecmwf.int/display/ECC/Python%203%20interface%20for%20ecCodes

Airbus. (2021). *Airbus global market forecast 2021-2040*.

Anand, H. & Moorhead, R. (1995). *A Unified Air-Sea Visualization System: Survey on Gridding Structures*. MISSISSIPPI STATE UNIV STENNIS SPACE CENTER CENTER FOR AIR SEA TECHNOLOGY.

Baldauf, M., Seifert, A., Förstner, J., Majewski, D., Raschendorfer, M. & Reinhardt, T. (2011). Operational Convective-Scale Numerical Weather Prediction with the COSMO Model: Description and Sensitivities. *Monthly Weather Review*, *139*(12), 3887–3905. https://doi.org/https://doi.org/10.1175/MWR-D-10-05013.1

Barleben, A., Haussler, S., Müller, R. & Jerg, M. (2020). A Novel Approach for Satellite-Based Turbulence Nowcasting for Aviation. *Remote sensing (Basel, Switzerland)*, *12*(14), 2255–. https://doi.org/10.3390/rs12142255

Battan, L. (1990). *Radar in meteorology : Battan Memorial and 40th Anniversary Radar Meteorology Conference*. American Meteorological Society.

Bauer, P., Thorpe, A. & Brunet, G. (2015). The quiet revolution of numerical weather prediction. *Nature*, *525*(7567), 47–55. https://doi.org/10.1038/nature14956

Bauer, T. & König, R. (2016). Lärmminderung im Landeanflug durch Anpassung des Höhen- und Geschwindigkeitsprofils. *Deutscher Luft- und Raumfahrtkongress 2016*.

Bénard, P. & Glinton, M. R. (2019). Circumventing the pole problem of reduced lat–lon grids with local schemes. Part I: Analysis and model formulation. *Quarterly Journal of the Royal Meteorological Society*, *145*(721), 1377–1391. https://doi.org/10.1002/qj.3509

Bouttier, F. (2001). The use of profiler data at ECMWF. *Meteorologische Zeitschrift*, *10*(6), 497–510. https://doi.org/10.1127/0941-2948/2001/0010-0497

Boyd, D. D. (2017). A Review of General Aviation Safety (1984–2017). *Aerospace Medicine and Human Performance*, *88*(7), 657–664. https://doi.org/10.3357/amhp.4862.2017

Brdar, S., Baldauf, M., Dedner, A. & Klöfkorn, R. (2012). Comparison of dynamical cores for NWP models: comparison of COSMO and Dune. *Theoretical and Computational Fluid Dynamics*, *27*(3-4), 453–472. https://doi.org/10.1007/s00162-012-0264-z

Buckeridge, S. & Scheichl, R. (2010). Parallel geometric multigrid for global weather prediction. *Numerical Linear Algebra with Applications*, *17*(2-3), 325–342. https://doi.org/10.1002/nla.699

Chaudhari, H., Lee, K. & Oh, J. (2007). Weather Prediction and computational aspects of Icosahedral-Hexagonal Gridpoint Model GME. *Parallel Computational Fluid Dynamics 2006* (S. 223–230). Elsevier. https://doi.org/10.1016/b978-044453035-6/50030-4

Cogato, A., Meggio, F., Migliorati, M. D. A. & Marinello, F. (2019). Extreme Weather Events in Agriculture: A Systematic Review. *Sustainability*, *11*(9), 2547. https://doi.org/10.3390/su11092547

COMET. (2015). *NWP Essentials: Structure and Dynamics*. Verfügbar 8. Oktober 2021 unter https://lyqx.sinaapp.com/wfiles/mstructure_dynamics/index.htm

COSMO. (2022). *Consortium for Small-scale Modelling*. Verfügbar 28. Januar 2022 unter http://www.cosmo-model.org/

Doviak, R., Zrnic, D. & Sirmans, D. (1979). Doppler weather radar. *Proceedings of the IEEE*, *67*(11), 1522–1553. https://doi.org/10.1109/proc.1979.11511

DWD. *promet - Probabilistische Wettervorhersage* [Jahrgang 37, Heft 3/4]. Jahrgang 37, Heft 3/4. http://www.dwd.de/promet, 2011. http://www.dwd.de/promet

DWD. (2020). *Open-Data-Server des Deutschen Wetterdienstes (DWD)*. Verfügbar 11. Januar 2022 unter https://opendata.dwd.de/

DWD. (2021a, 8. Oktober). *Aufgaben des DWD*. Verfügbar 8. Oktober 2021 unter https://www.dwd.de/DE/derdwd/aufgaben/aufgaben_node.html

DWD. (2021b, 8. Oktober). *Datenassimilation am DWD*. Verfügbar 8. Oktober 2021 unter https://www.dwd.de/DE/forschung/wettervorhersage/num_modellierung/02_datenassimilation/datenassimilation_node.html

DWD. (2021c, 8. Oktober). *ICON des DWD*. Verfügbar 8. Oktober 2021 unter https://www.dwd.de/DE/forschung/wettervorhersage/num_modellierung/01_num_vorhersagemodelle/icon_beschreibung.html?nn=19880

EASA & EAA. (2019). *European aviation environmental report 2019.* Publications Office. https://doi.org/10.2822/309946

ECMWF. (2022). *How to calculate wind speed and wind direction from u and v components of the wind?* Verfügbar 18. Januar 2022 unter https://confluence.ecmwf.int/pages/viewpage.action?pageId=133262398

Eyre, J. R., Bell, W., Cotton, J., English, S. J., Forsythe, M., Healy, S. B. & Pavelin, E. G. (2021). Assimilation of satellite data in numerical weather prediction. Part II : recent years. *Quarterly Journal of the Royal Meteorological Society*. https://doi.org/10.1002/qj.4228

Eyre, J. R., English, S. J. & Forsythe, M. (2019). Assimilation of satellite data in numerical weather prediction. Part I: The early years. *Quarterly Journal of the Royal Meteorological Society*, *146*(726), 49–68. https://doi.org/10.1002/qj.3654

Folini, D. (2018). Climate, weather, space weather: model development in an operational context. *Journal of space weather and space climate*, *8*, A32–. https://doi.org/10.1051/swsc/2018021

Fucile, E. (2021). *GRIB-API migration* (ECMWF, Hrsg.). Verfügbar 30. Dezember 2021 unter https://confluence.ecmwf.int/display/ECC/GRIB-API%20migration

GitHub. (2022). *GitHub start*. Verfügbar 11. Januar 2022 unter https://github.com/

Glahn, H. R. & Lowry, D. A. (1972). The Use of Model Output Statistics (MOS) in Objective Weather Forecasting. *11*(8), 1203–1211. https://doi.org/10.1175/1520-0450(1972)011<1203:tuomos>2.0.co;2

Gneiting, T. & Raftery, A. E. (2005). Weather Forecasting with Ensemble Methods. *Science*, *310*(5746), 248–249. https://doi.org/10.1126/science.1115255

Gneiting, T., Raftery, A. E., Westveld, A. H. & Goldman, T. (2005). Calibrated Probabilistic Forecasting Using Ensemble Model Output Statistics and Minimum CRPS Estimation. *133*(5), 1098–1118. https://doi.org/10.1175/mwr2904.1

Gorelick, M. & Ozsvald, I. (2020, 19. Mai). *High Performance Python*. O'Reilly UK Ltd. https://www.ebook.de/de/product/37935143/micha_gorelick_ian_ozsvald_high_performance_python.html

Griefahn, B., Kaniowski, A. M., Klepper, G., Lingner, S. & Steinebach, G. (2006, 2. August). *Leben mit Lärm? Risikobeurteilung und Regulation des Umgebungslärms im Verkehrsbereich*. Springer Berlin Heidelberg. https://www.ebook.de/de/product/5461923/b_griefahn_a_m_kaniowski_g_klepper_s_lingner_g_steinebach_leben_mit_laerm.html

Gultepe, I., Sharman, R., Williams, P. D., Zhou, B., Ellrod, G., Minnis, P., Trier, S., Griffin, S., Yum, S. S., Gharabaghi, B., Feltz, W., Temimi, M., Pu, Z., Storer, L. N., Kneringer, P., Weston, M. J., Chuang, H.-y., Thobois, L., Dimri, A. P., … Neto, F. L. A. (2019). A Review of High Impact Weather for Aviation Meteorology. *Pure and Applied Geophysics*, *176*(5), 1869–1921. https://doi.org/10.1007/s00024-019-02168-6

Gupta, M. (2019). *Understanding different Box Plot with visualization* (GeeksforGeeks, Hrsg.). Verfügbar 30. Januar 2022 unter https://www.geeksforgeeks.org/understanding-different-box-plot-with-visualization/

Gustafsson, N., Janjić, T., Schraff, C., Leuenberger, D., Weissmann, M., Reich, H., Brousseau, P., Montmerle, T., Wattrelot, E., Bučánek, A., Mile, M., Hamdi, R., Lindskog, M., Barkmeijer, J., Dahlbom, M., Macpherson, B., Ballard, S., Inverarity, G., Carley, J., … Fujita, T. (2018). Survey of data assimilation methods for convective-scale numerical weather prediction at operational centres. *Quarterly Journal of the Royal Meteorological Society*, *144*(713), 1218–1256. https://doi.org/10.1002/qj.3179

Haupt, S. E., Chapman, W., Adams, S. V., Kirkwood, C., Hosking, J. S., Robinson, N. H., Lerch, S. & Subramanian, A. C. (2021). Towards implementing artificial intelligence post-processing in weather and climate: proposed actions from the Oxford 2019 workshop. *379*(2194), 20200091. https://doi.org/10.1098/rsta.2020.0091

Hocking, W. K., Carey-Smith, T., Tarasick, D. W., Argall, P. S., Strong, K., Rochon, Y., Zawadzki, I. & Taylor, P. A. (2007). Detection of stratospheric ozone intrusions by windprofiler radars. *Nature*, *450*(7167), 281–284. https://doi.org/10.1038/nature06312

Jäger, D., Zellmann, C., Wunderli, J., Scholz, M., Abdelmoula, F. & Gerber, M. (2021). Validation of an airline pilot assistant system for low-noise approach procedures. *Transportation Research Part D: Transport and Environment*, *99*, 103020. https://doi.org/10.1016/j.trd.2021.103020

JetBrains. (2021a). *Enabling Git as Version Control*. Verfügbar 12. November 2021 unter https://www.jetbrains.com/help/pycharm/2021.1/using-git-integration.html

JetBrains. (2021b). *Optimize your code using profilers*. Verfügbar 25. November 2021 unter https://www.jetbrains.com/help/pycharm/profiler.html

JetBrains. (2022). *Herunterladen PyCharm*. Verfügbar 11. Januar 2022 unter https://www.jetbrains.com/de-de/pycharm/download/#section=windows

Karateev, P. & Karpov, S. (2021). *PyCharm no WSL interpreter Workaround* (Y. by JetBrains, Hrsg.). Verfügbar 9. November 2021 unter https://youtrack.jetbrains.com/issue/PY-50802

Kiktev, D., Joe, P., Isaac, G. A., Montani, A., Frogner, I.-L., Nurmi, P., Bica, B., Milbrandt, J., Tsyrulnikov, M., Astakhova, E., Bundel, A., Bélair, S., Pyle, M., Muravyev, A., Rivin, G., Rozinkina, I., Paccagnella, T., Wang, Y., Reid, J., … Ahn, K.-D. (2017). FROST-2014: The Sochi Winter Olympics International Project. *Bulletin of the American Meteorological Society*, *98*(9), 1908–1929. https://doi.org/10.1175/bams-d-15-00307.1

Kühne, C. G., Scholz, M. & Abdelmoula, F. (2018). *LNAS - A PILOT ASSISTANCE SYSTEM FOR ENERGY-OPTIMAL APPROACHES USING EXISTING AIRCRAFT INFRASTRUCTURE* (Techn. Ber.). AEGATS'18. Toulouse, France.

Lorenc, A. C. (1981). A Global Three-Dimensional Multivariate Statistical Interpolation Scheme. *109*(4), 701–721. https://doi.org/10.1175/1520-0493(1981)109<0701:agtdms>2.0.co;2

Lynch, P. (2006, 1. November). *The Emergence of Numerical Weather Prediction: Richardson's Dream*. Cambridge University Press. https://www.ebook.de/de/product/5873603/peter_lynch_the_emergence_of_numerical_weather_prediction_richardson_s_dream.html

Majewski, D., Liermann, D., Prohl, P., Ritter, B., Buchhold, M., Hanisch, T., Paul, G., Wergen, W. & Baumgardner, J. (2002). The Operational Global Icosahedral–Hexagonal Gridpoint Model GME: Description and High-Resolution Tests. *Monthly Weather Review*, *130*(2), 319–338. https://doi.org/10.1175/1520-0493(2002)130<0319:togihg>2.0.co;2

MathWorks. (2021). *MATLAB: products*. Verfügbar 29. Dezember 2021 unter https://ch.mathworks.com/de/products/matlab.html

MeteoSchweiz. (2021). *Internationale Zusammenarbeit: COSMO*. Verfügbar 1. Dezember 2014 unter https://www.meteoschweiz.admin.ch/home/forschung-und-zusammenarbeit/internationale-zusammenarbeit/cosmo.html

Moninger, W. R., Mamrosh, R. D. & Pauley, P. M. (2003). Automated Meteorological Reports from Commercial Aircraft. *Bulletin of the American Meteorological Society*, *84*(2), 203–216. https://doi.org/10.1175/bams-84-2-203

Neiburger, M. & Wexler, H. (1961). Weather Satellites. *205*, 80–94. https://doi.org/10.1038/scientificamerican0761-80

Oeser, F. (2021). *Tessellation einer Kugeloberfläche mit Dreiecken*. Verfügbar 30. Dezember 2021 unter http://www.florian-oeser.de/rtr/ue2/

Petersen, R. A. (2016). On the Impact and Benefits of AMDAR Observations in Operational Forecasting—Part I: A Review of the Impact of Automated Aircraft Wind and Temperature Reports. *Bulletin of the American Meteorological Society*, *97*(4), 585–602. https://doi.org/10.1175/bams-d-14-00055.1

Python. (2021). *Python Source Releases* (P. S. Foundation, Hrsg.). Verfügbar 29. Dezember 2021 unter https://www.python.org/downloads/

Reinert, D., Prill, F., Frank, H., Denhard, M., Baldauf, M., Schra, C., Gebhardt, C., Marsigli, C. & Zängl, G. (2021). *DWD Database Reference for the Global and Regional ICON and ICON-EPS Forecasting System* (2.1.6).

ResearchGate. (2021). *C-grid staggering*. Verfügbar 30. Dezember 2021 unter https://www.researchgate.net/figure/Arakawa-C-grid-staggering-Source-WRF-NMM-Core-Skamarock-and-Dudhia-2010-The_fig2_262972565

Richardson, L. F. (1922). *Weather Prediction by Numerical Process*. Cambridge Univerity Press.

Rieger, D. (2021). *Priority Project "C2I", Transition of COSMO to ICON-LAM* (COSMO, Hrsg.). Verfügbar 26. August 2021 unter http://www.cosmo-model.org/content/tasks/priorityProjects/c2i/default.htm

Rinehart, R. E. & Garvey, E. T. (1978). Three-dimensional storm motion detection by conventional weather radar. *Nature*, *273*(5660), 287–289. https://doi.org/10.1038/273287a0

Santoalla, D. V. (2021). *ecCodes installation* (ECMWF, Hrsg.). Verfügbar 30. Dezember 2021 unter https://confluence.ecmwf.int/display/ECC/ecCodes%20installation

Satoh, M., Matsuno, T., Tomita, H., Miura, H., Nasuno, T. & Iga, S. (2008). Nonhydrostatic icosahedral atmospheric model (NICAM) for global cloud resolving simulations. *Journal of Computational Physics*, *227*(7), 3486–3514. https://doi.org/https://doi.org/10.1016/j.jcp.2007.02.006

Schär, C., Fuhrer, O., Arteaga, A., Ban, N., Charpilloz, C., Girolamo, S. D., Hentgen, L., Hoefler, T., Lapillonne, X., Leutwyler, D., Osterried, K., Panosetti, D., Rüdisühli, S., Schlemmer, L., Schulthess, T. C., Sprenger, M., Ubbiali, S. & Wernli, H. (2020). Kilometer-Scale Climate Models: Prospects and Challenges. *Bulletin of the American Meteorological Society*, *101*(5), E567–E587. https://doi.org/10.1175/bams-d-18-0167.1

Schröder, D. (2014). GRIB2 im DWD: aus Vortrag Kundenforum DWD am 28.04.2014 (DWD, Hrsg.). Verfügbar 6. Oktober 2021 unter https://www.dwd.de/DE/leistungen/opendata/help/modelle/grib2_erlaeuterungen.pdf?__blob=publicationFile&v=1

Seward, J. & Blättermann, M. (2019). *Ubuntu Manpage Repository* (C. Ltd., Hrsg.). Verfügbar 11. Januar 2022 unter http://manpages.ubuntu.com/manpages/bionic/de/man1/bzip2.1.html

Simmons, A. J. & Hollingsworth, A. (2002). Some aspects of the improvement in skill of numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, *128*(580), 647–677. https://doi.org/10.1256/003590002321042135

Staffell, I. & Pfenninger, S. (2018). The increasing impact of weather on electricity supply and demand. *Energy*, *145*, 65–78. https://doi.org/10.1016/j.energy.2017.12.051

Theis, S., Gebhart, C. & Bouallègue, Z. B. (2015). *Beschreibung des COSMO-DE-EPS und seiner Ausgabe in die Datenbanken des DWD* (Techn. Ber.) [Version 2.0, 22.09.2015]. DWD.

Tomita, H. & Satoh, M. (2004). A new dynamical framework of nonhydrostatic global model using the icosahedral grid. *Fluid Dynamics Research*, *34*(6), 357–400. https://doi.org/https://doi.org/10.1016/j.fluiddyn.2004.03.003

Trub, R., Moser, D., Schafer, M., Pinheiro, R. & Lenders, V. (2018). *Monitoring Meteorological Parameters with Crowdsourced Air Traffic Control Data*. IEEE. https://doi.org/10.1109/ipsn.2018.00010

Velden, C., Simpson, J., Liu, W. T., Hawkins, J., Brueske, K. & Athens, R. (2003). The Burgeoning Role of Weather Satellites, 217–247. https://doi.org/10.1029/sp055p0217

WSL. (2021). *Install WSL* (Microsoft, Hrsg.). Verfügbar 29. Dezember 2021 unter https://docs.microsoft.com/en-us/windows/wsl/install

Zängl, G., Reinert, D., Rípodas, P. & Baldauf, M. (2015). The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. *Quarterly journal of the Royal Meteorological Society*, *141*(687), 563–579. https://doi.org/10.1002/qj.2378