Masterarbeit

ELEKTROCHEMISCHE CHARAKTERISIERUNG VON ELEKTROLYSE STACKS MITTELS IMPEDANZSPEKTROSKOPIE

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Clemens Weiskopf, B.Sc., M.Sc.
51815266

betreut von
Dr.rer.nat Stefan Pofahl

begutachtet von
DI (FH) Gernot Hofer

Graz, Juli 2023

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....
Unterschrift

DANKSAGUNG

Zuallererst möchte ich mich bei der HyCentA Research GmbH und dabei namentlich bei Ass.Prof. Dipl.-Ing. Dr.techn. Alexander Trattner und Dipl.-Ing. Dr. techn. Marie-Gabrielle Macherhammer für das Ermöglichen dieser Arbeit und den fachlichen Input bedanken.

Ein besonderer Dank gilt Dr.rer.nat Stefan Pofahl, der die Betreuung übernommen hat und mir immer mit Rat und Tat zur Seite stand und somit maßgeblich zum Gelingen dieser Arbeit beigetragen hat. Selbes gilt für Dipl.-Ing. Joshua Eder, B.Sc., dem ich überdies noch für die Unterstützung in den schwierigen Phasen dieser Arbeit danken möchte.

Bedanken möchte ich mich auch bei meinem FH-Betreuer DI (FH) Gernot Hofer für seine Unterstützung und das Feedback.

Zuletzt möchte ich mich noch bei meiner Familie sowie meinen Freunden, die mich in dieser intensiven Zeit unterstützt haben, bedanken.

„Es ist erstaunlich, dass trotz aller Siege der Technik die Natur noch immer am Boden liegt.“
Ernst R. Hauschka (1926 - 2012)

KURZFASSUNG

Angesichts der Energiekrise wird Power-to-Gas als einzige praktische Lösung für die saisonübergreifende Langzeitspeicherung von Energie und als Antwort auf die schwankende und wachsende Stromerzeugung aus erneuerbaren Energien angesehen. Die Grundlage dieser Technologie ist die Erzeugung von grünem Wasserstoff durch Elektrolyse von Wasser mittels Elektrolyse Stacks, die aus mehreren Zellen bestehen. Bei der Entwicklung dieser Zellen wird die elektrochemische Impedanzspektroskopie (EIS) als zerstörungsfreie Methode zur Überwachung des Betriebszustandes eingesetzt. Diese Methode erlaubt es, bestimmte Phänomene von Aktivierungs- und Degradationsprozessen nicht nur einer bestimmten Zelle, sondern auch den Zellkomponenten zuzuordnen. Über ein Ersatzschaltbild können die Veränderungen empirisch über die Komponentenwerte ermittelt und dokumentiert werden. Da EIS-Messungen meist an Einzelzellen beziehungsweise kleinen Stapeln erfolgen, sind bis auf wenige Ausnahmen keine Geräte zur Messung von Stapeln mit einer Stromaufnahme größer als 30 A am Markt erhältlich. Um diese Einschränkung zu umgehen, wurde im Zuge des EU-Projektes Recycalyze (ID: 861960) von der HyCentA Research GmbH ein eigener Aufbau zur Gewinnung der Messdaten entwickelt. Das Ziel dieser Masterarbeit ist die Datenverarbeitung und Auswertung dieser Messungen. Dazu wird die Frequenzanalyse hinsichtlich der Genauigkeit in Abhängigkeit von der Messdauer und Phänomenen wie spektralen Leckagen untersucht. Außerdem wird der Einfluss verschiedener Fenster verglichen und bewertet. Um die Auswirkungen auf die Membranelektrodenanordnung (MEA) zu zeigen, werden einfache Ersatzschaltbilder simuliert und gezeigt. Die Veränderung der Bauteilwerte kann bestimmte Effekte zeigen, die im Nyquist Plot aufgezeigt und mit gemessenen Impedanzkurven verglichen werden können. Die Frequenzanalyse, sowie die grafische Aufbereitung und Archivierung erfolgt in der Programmiersprache JULIA. Zur Validierung des Messaufbaus und der Datenauswertung wurde eine Messung mit einem marktüblichen Produkt, bei geringer Stromdichte, erfolgreich durchgeführt. Mit diesem Messaufbau und der Datenauswertung kann die HyCentA Research GmbH Messungen in einem Leistungsbereich anbieten, die bisher nicht möglich waren.

ABSTRACT

As the energy crisis evolves, Power-to-Gas is seen as the only practical solution for cross-seasonal long-time storage of energy and the answer to the fluctuating and growing renewable power generation. The foundation of this technology is the production of green hydrogen by electrolysis of water via electrolysis stacks which consist of multiple cells. For the development of these cells, electrochemical impedance spectroscopy (EIS) is used as a nondestructive method to monitor the condition in operation. This method allows to identify and assign certain phenomena of activation and degradation processes not only to cells but to the cell components. By means of fitting an equivalent electric circuit these changes can be determined and documented empirically by the component values. Since this is mostly done on single cells respectively small stacks, there are as of yet no devices on the market to measure stacks with a current consumption bigger than 30 A. Consequently, in order to bypass this restriction, HyCentA Research GmbH developed its own setup for obtaining the measurement data in the course of the EU project Recycalyze. The objective of this master's thesis is the data processing and analysing of this measurements. Therefore, the frequency analysis is investigated concerning accuracy depending on the measurement duration and phenomena like spectral leakage. Additionally, the influence of different windows is compared and evaluated. To show the effects of change in the membrane electrode assembly (MEA), simple equivalent circuits are shown and simulated. Changing the component values can show certain effects, which can be pointed out in the Nyquist plot and compared to measured impedance curves. The frequency analysis, the graphical preparation and archiving is done in the programming language JULIA. To validate the measurement setup and data analysis, a measurment with a market available product was successfully done, at low current density. With this measurement setup and data evaluation the HyCentA Research GmbH can offer measurements in a power range which has not been possible up to now.

INHALTSVERZEICHNIS

1 EINLEITUNG

Was wir durch die Temperaturrekorde [1] nun deutlicher denn je zu spüren bekommen, sind die Auswirkungen des Klimawandels, der schon seit Jahrzehnten von Wissenschaftlern prognostiziert und von der Politik und Lobbyverbänden sträflich vernachlässigt und großteils weiterhin ignoriert wird. Die Wetterextreme häufen sich und werden nicht zuletzt durch die immer noch steigenden Emissionen weiter befeuert. Nicht zuletzt der Krieg in der Ukraine hat uns deutlich vor Augen geführt, wie abhängig wir von fossilen Energieträgern sind und wie unmittelbar die Dringlichkeit der Energiewende für den Erhalt eines lebenswerten Planeten ist.

Das Erreichen des Erdüberlastungstags, [2] jener Tag eines Jahres an dem nachhaltig nutzbare Ressourcen verbraucht sind, im Juli 2022 zeigt deutlich wie weit wir noch von einer Energieautonomie entfernt sind. Auch die Statistik über die weltweiten CO_2 - Emissionen nach Sektoren in Abbildung 1.1 verdeutlicht, dass in der Energieindustrie noch eine überwiegende Abhängigkeit von nicht erneuerbaren Energien besteht. Es lässt sich auch kein signifikanter Trend in Richtung einer Reduktion über die Jahre hinweg feststellen.

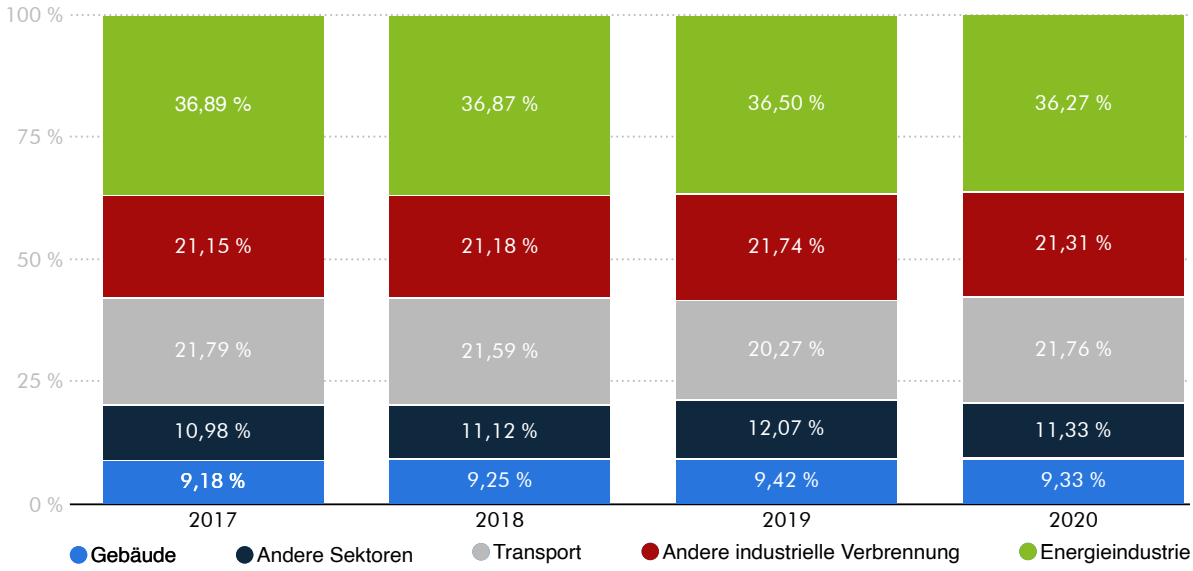

Abbildung 1.1: Verteilung der CO_2 - Emissionen weltweit nach Sektoren bis 2020. [3]

Dies gilt auch für die anderen Sektoren, deren prozentualer Ausstoß keine markante Veränderung erkennen lässt. Ein Fünftel kann auf den Transport zurückgeführt werden, womit auch diesem Sektor ein großer Umbruch bevorsteht. Dies ist dahingehend relevant, da die Menschen die notwendige Veränderung unmittelbar in ihrem Alltag, im Gegensatz zur industriellen Nutzung und Energiegewinnung, bemerken werden.

Wenn man die Emissionen von Österreich betrachtet, zeigt sich ein ähnliches Bild. Der größte Anteil mit 44 % ist auf die Industrie zurückzuführen [4]. Am Beispiel der Voest Stahlwerke lässt sich die Dimension der bevorstehenden Aufgaben veranschaulichen. Diese haben einen Energieverbrauch von ca. 40 TW h pro

1 Vgl. GeoSphere Austria – Bundesanstalt für Geologie, Geophysik, Klimatologie und Meteorologie 2022.
2 Vgl. Umweltbundesamt 2022.
3 Vgl. Statista 2022.
4 Vgl. Umweltbundesamt 2021.

Jahr, der nur zu 5,3 % mit Strom gedeckt wird [5]. Der Rest wird von den fossilen Energieträgern Erdgas, Koks und Kohle bereitgestellt. Die ca. 1300 Windkraftanlagen, mit einer Gesamtnennleistung von 3300 MW, die in Österreich derzeit installiert sind, haben im Jahr 2021 8,5 TW h Strom produziert [6]. Das heißt, die Anzahl der Windkraftanlagen müsste in Österreich verfünffacht werden, die Transport- und Umwandlungsverluste nicht mit eingerechnet, um den Energiebedarf dieses Unternehmens nachhaltig zu decken. Auch der Verkehrssektor hat mit 30 % ein enormes Emissionsproblem. Durch die Elektrifizierung der PKW-Flotte nimmt zwar der Gesamtenergieaufwand ab, jedoch verlagert sich dieser auf das Stromnetz. Wenn man sich vor Augen führt, dass laut Verkehrsclub Österreich [7] ein Windrad ca. 3000 E-Autos versorgen kann, kommen bei ungefähr 5,1 Mio. zugelassenen Fahrzeugen [8] nochmals 1700 Windräder hinzu, würden diese rein über das öffentliche Stromnetz geladen. Der dritte große Emittent sind Gebäude, die vor allem in den Großstädten mit Gas geheizt werden und für 10 % der Gesamtemissionen verantwortlich sind. Neben Fernwärme und Biomasse steht hier vor allem die Wärmepumpe im Fokus. Da diese mit Strom betrieben wird, kommt ein weiterer Verbraucher im Stromnetz hinzu. Dieser erhöhte Verbrauch steht der geringeren Produktion der Photovoltaik im Winter und in der Nacht gegenüber. Bedenkt man dann noch, dass nicht nur die Photovoltaik, sondern auch die Windkraft in ihrer Erzeugungsleistung fluktuieren und an günstigen Tagen teilweise jetzt schon gedrosselt werden müssen, stößt man unweigerlich auf eines der wichtigsten Puzzlestücke der Energiewende, die Energiespeicher.

1.1 Energiespeicher

Ein Speicherzyklus besteht immer aus den drei Schritten: Einspeichern (Laden), Speichern (Halten) und Ausspeichern (Entladen). Diese sind nötig, weil elektrische Energie nicht praktikabel speicherbar ist und dazu immer in eine andere Energieform umgewandelt werden muss [9]. Energiespeicher können grundsätzlich nach ihren technischen Eigenschaften klassifiziert werden. Sie sind jedoch schwer miteinander zu vergleichen, da sie auf unterschiedlichen physikalischen Prinzipien beruhen. So gibt es mechanische Speicher, die durch Masse auf potenzielle oder kinetische Energie setzen, wie etwa Pumpspeicherkraftwerke oder Schwungmassenspeicher. Hierbei muss die elektrische Energie in mechanische Energie umgewandelt und beim Ausspeichern wieder zurückgewandelt werden. Konventionelle Energieträger wie Erdgas und Kohle speichern die Energie in Form von chemischen Bindungen. Elektrische Energie kann direkt mit Elektronikbauteilen wie Kondensatoren und Spulen gespeichert werden, jedoch sind diese Speicher in ihrer Größe und Speicherfähigkeit stark beschränkt. Batterien gehen einen Mittelweg, da sie die elektrische und chemische Speicherung kombinieren. Thermische Speicher nutzen Temperaturunterschiede, in Phasenübergängen und chemischen Bindungen. [10]

In Abbildung 1.2 ist die Speicherkapazität über die Ausspeicherzeit, im doppelt logarithmischen Ragone-Diagramm [11], aufgetragen. Hierbei werden die Speicher einem Wirkprinzip zugeordnet und in Punktewolken im Diagramm aufgetragen. So kann das Einsatzgebiet der jeweiligen Technologie veranschaulicht werden. Zusätzlich sind noch die Jahresstromverbräuche, in den vertikal gestrichelten Linien, vom Haushalt bis zur Großstadt eingetragen, um die Speicherkapazität in eine erfassbare Relation zu setzten.

Die Speicher können so in Lang- und Kurzzeitspeicher unterschieden werden, wobei die Grenze bei 24 h

[5] Vgl. Voestalpine 2021.
[6] Vgl. IG-Windkraft 2022.
[7] Vgl. VCÖ 2020.
[8] Vgl. Statistik Austria 2022.
[9] Vgl. Michael Sterner, Ingo Stadler 2017, S. 26.
[10] Vgl. Michael Sterner, Ingo Stadler 2017, S. 647.
[11] Vgl. V. Ragone 1968.

liegt. Es wird gezeigt, dass elektrische Bauteile und Schwungmassenspeicher nur für sehr kurze Speicherzeiträume eine Eignung besitzen und somit bei der saisonalen Energiespeicherung keine Rolle spielen. Auch Batterien und Fernwärme sind nur bedingt zur Speicherung über einen längeren Zeitraum fähig.

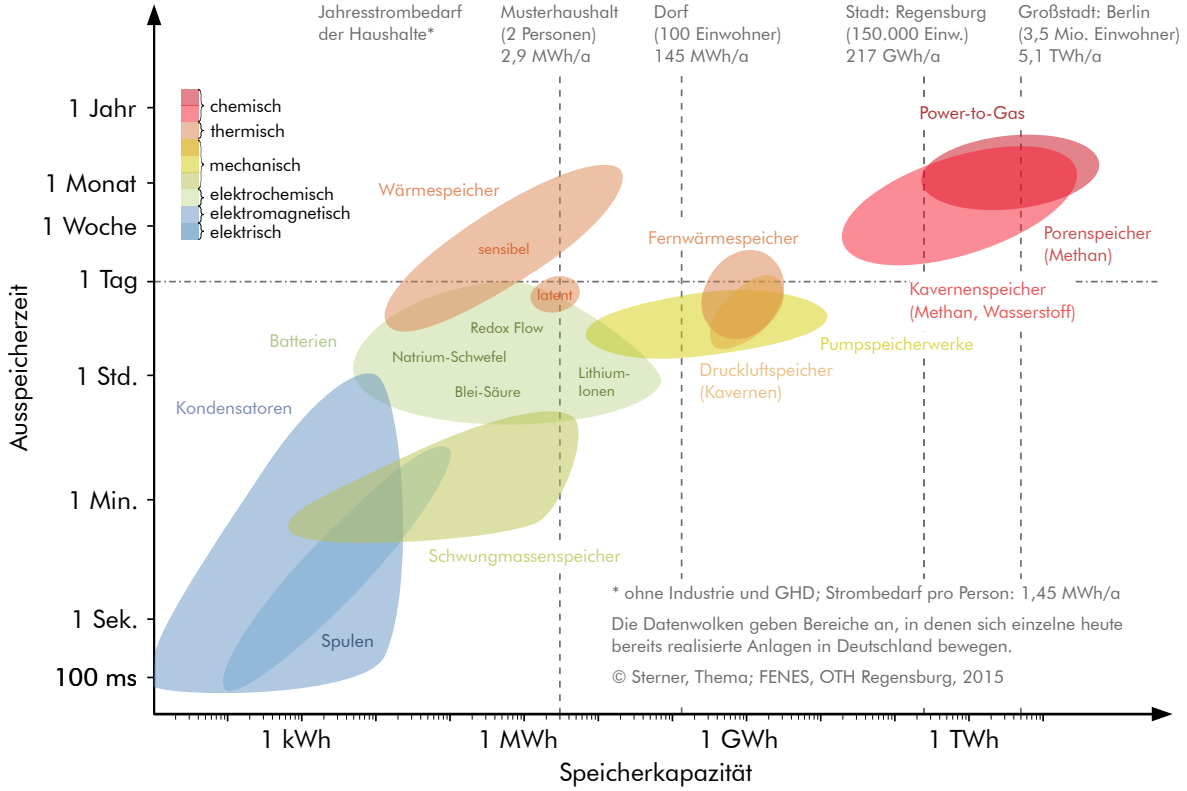

Abbildung 1.2: Ausspeicherzeit verschiedener Energiespeicher über deren Speicherkapazität und sortiert nach Technologien, für in Deutschland 2016 existierende Anlagen. [12]

Nur die chemische Speicherung durch Umwandlung zu Gas, auch als Power-to-Gas bekannt, stellt einen praktikablen Lösungsansatz dar. So können sehr hohe Energiemengen über einen langen Zeitraum gespeichert werden, womit eine Verlagerung des Energieüberschusses aus dem Sommer in den Winter möglich ist. Durch die synthetische Herstellung von Methan kann die bereits bestehende Gasinfrastruktur für die Verteilung genutzt werden. Hierbei ist vor allem die hohe Transportkapazität einer Gaspipeline vorteilhaft, sowie die bereits bestehende Speicherkapazität der Gasinfrastruktur. Außerdem kann dieses synthetisch hergestellte Gas auch für industrielle Prozesse verwendet werden, wobei diese Methode im besten Fall CO_2-neutral jedoch nicht CO_2-frei ist. Die Basis für die Herstellung dieser synthetischen Kohlenwasserstoffe ist grüner Wasserstoff, wie in Kapitel 3.3 näher beschrieben, wobei dieser auch direkt ohne weitere Umwandlungsschritte gespeichert und industriell verwendet werden kann. Somit stellt Wasserstoff das zukünftige Rückgrat der Langzeitspeicherung dar und wird auch in vielen anderen industriellen Prozessen zunehmend an Relevanz gewinnen.

1.2 Wasserstoffwirtschaft

Der Wechsel hin zu einer nachhaltigen Gesellschaft deren Energiebedarf zu 100 % aus erneuerbaren Energien wie Wind, Solar, Erdwärme oder Biomasse gedeckt ist, beinhaltet auch den Schritt hin zu einer Wasserstoffwirtschaft. Dadurch eröffnet sich ein Energiemarkt, an dem jedes Land teilhaben kann, da dieser

[12] Adaptiert von: Michael Sterner, Ingo Stadler 2017, S. 654.

nicht mehr auf der Ausbeutung von Bodenschätzen, die weltweit ungleich verteilt sind, beruht. Somit geht es um nicht weniger als die nächste industrielle Revolution. Aufgrund der vielfältigen Einsatzmöglichkeiten von Wasserstoff kann dieser in der Industrie, Mobilität, den Privathaushalten, sowie für dezentralisierte Versorgung genutzt werden, womit dieser eine Schlüsselfunktion in der Dekarbonisierung spielt. Dies heißt aber auch, dass die Anzahl an elektrochemischen Maschinen rasant ansteigen muss. Vor allem in Industrieprozessen wie etwa der Düngemittelherstellung, Halbleiterproduktion, Lebensmittelverarbeitung, Wasseraufbereitung und der Metallurgie werden große Mengen von grünem Wasserstoff gebraucht. Hier hat die Umstellung einen besonders großen Einfluss, da von diesen Anlagen ein Großteil der Emissionen in diesem Sektor produziert werden. [13]

Neben Batterien und Brennstoffzellen bilden vor allem die Elektrolyseure die Basis des Wasserstoff-Ökosystems, da mit ihnen der Wasserstoff für all diese Einsatzgebiete produziert wird. Somit stehen diese zwischen der Produktion und der Nutzung in der Wertschöpfungskette und spielen eine zentrale Rolle.

1.3 Motivation

Da in Zukunft die Elektrolyseanlagen in ihrer Größe und damit ihrer Leistung immer weiter ansteigen werden [14], ist es von hoher Bedeutung, den Zustand der einzelnen Zellen zu überwachen. Derzeit erfolgt das Monitoring über Parameter wie etwa der Wärmeentwicklung und damit dem Effizienzgrad der Anlage, aber auch die Zellspannung wird mittels Cell Voltage Monitoring (CVM) gemessen. Damit ist eine Einschätzung des Zellzustandes möglich, jedoch können nur Zellen miteinander verglichen werden und nicht die einzelnen Komponenten einer Zelle. Der Zustand der Einzelzellen kann im eingebauten Zustand jedoch nicht beurteilt werden. Hierfür müsste der Elektrolyse-Stack in seine Komponenten zerlegt und einzeln geprüft werden. Mit der Impedanzspektroskopie können Rückschlüsse auf den Zustand der Zellen gezogen werden und dies, während die Anlage in Betrieb ist. Deshalb ist dieses Verfahren nicht nur in der Forschung, sondern auch für den kommerziellen Einsatz bei der kontinuierlichen Zustandsüberwachung von besonderem Interesse. Es gibt vielfältige Anwendungsmöglichkeiten, die in Kapitel 5 benannt werden.

Derzeit kommt das Verfahren hauptsächlich in der Materialforschung zum Einsatz, um verschiedene Materialien beziehungsweise Materialkombinationen miteinander zu vergleichen. Aber auch der Vergleich von verschiedenen Stacks und deren Bauformen ist damit möglich. Dabei ist besonders der Hochfrequenzwiderstand, näher beschrieben in Kapitel 5, ein wichtiger Kennwert, der den Zustand und die Degradation der Membran beschreibt. Außerdem können mit dieser Analysemethode nicht nur Materialvergleiche gemacht werden, sondern auch die Charakterisierung der Komponenten nach der Assemblierung des Stacks. Hierbei handelt es sich um eine nachgelagerte Qualitätskontrolle, mit der Fertigungsverfahren überwacht und unterschiedliche Verfahren verglichen werden können. Ein Anwendungsfall, der besonders für den Betrieb einer Elektrolyseanlage von Bedeutung ist, ist die Untersuchung von Degradationsphänomenen aufgrund verschiedener Betriebszustände. Hierbei kann durch eine Impedanzspektroskopie vor und nach einem Betriebszustand dessen Auswirkung isoliert und analysiert werden. Dadurch können Szenarien identifiziert werden, die besonders schädlich für die Haltbarkeit des Systems sind.

Aufgrund des breiten Einsatzgebietes sind auf dem Markt Komplettsysteme, bestehend aus Messelektronik und Auswertesoftware, erhältlich. Diese sind jedoch nicht für den Leistungsbereich konzipiert, der für die Analyse von großen Stacks notwendig ist. Deshalb muss hier der Weg über den Eigenbau mit entsprechend leistungsfähigen Komponenten gegangen werden. Dies hat besonders in der Auswertung und der Flexibilität des Testablaufes einige Vorteile. Beispielsweise ist man bei der Einprägung des Signales

[13] Vgl. Alexander Trattner, Manfred Klell, Fabian Radner; 2022, S. 396.
[14] Vgl. Tom Smolinka, Martin Günther, Jürgen Garche; 2011, S. 38.

nicht mehr an die Grenze der Hersteller gebunden und kann über die Hardware frei verfügen. Auch die Signalform sowie die Dauer und Anzahl der Einprägefrequenzen unterliegt keiner Limitierung mehr.

Ein weiterer wichtiger Punkt ist die Anzahl der Messkanäle, die mit der verwendeten Messelektronik beliebig erweitert werden können und neben der Impedanzmessung auch noch andere Messgrößen aufzeichnen kann. Ein besonders wichtiger Punkt ist der Zugriff auf die Rohdaten der Messung und damit auch auf die Informationen im Zeitbereich. Dadurch ist es auch möglich, neue Auswerteverfahren auf alte Messungen anzuwenden, was durch die reine Ausgabe der Ergebnisse bei den industriellen Lösungen nicht mehr möglich wäre.

Außerdem werden die Ergebnisse von manchen Herstellern adaptiert, indem sie die Impedanzwerte einer Glättung unterziehen. Durch den Zugriff auf die Rohdaten besteht zusätzlich die Möglichkeit Algorithmen im Zeitbereich, wie etwa Mustererkennung, zu verwenden. Da auf die Auswertealgorithmen und die Ergebnisdaten zugegriffen werden kann, können Fehler im Messaufbau oder andere Messanwendungen sowie Ausgaben in die Analyse integriert beziehungsweise korrigiert werden.

2 AUFGABENSTELLUNG UND RANDBEDINGUNGEN

Diese Arbeit wurde von der HyCentA Research GmbH im Rahmen des F&E Projektes "Recycalyse" [15] ermöglicht. Diese hat sich ganz der Erforschung von Wasserstofftechnologien verschrieben von der Infrastruktur über die Mobilität bis hin zur Erzeugung. Letzteres ist auch der Bereich, dem diese Arbeit zuzuschreiben ist und im spezifischen der Analyse von Proton-Exchange-Membran (PEM) Elektorlyseuren. Die Wasserelektrolyse ist eine Schlüsseltechnologie für die Erzeugung von Wasserstoff und damit für die Speicherung von Energie aus erneuerbaren Energiequellen. Im Rahmen dieses Forschungsprojektes werden verschiedene Elektrolyse-Stacks ausgewertet und verglichen.

Die Hauptmethode zur Charakterisierung dieser Stacks (im deutschen Sprachraum oft auch als Stapel bezeichnet) ist die elektrochemische Impedanzspektroskopie (EIS). An der HyCentA GmbH wird derzeit eine eigene Toolchain aus spezifischen Hard- und Softwarewerkzeugen entwickelt, um die stapelspezifischen Informationen zu sammeln und zu analysieren. Dieser Ansatz eröffnet die Möglichkeit, neue interdisziplinäre Methoden für Algorithmen zur Signalanalyse im Zeitbereich zu nutzen. Die beiden Hauptaspekte dieser Masterarbeit sind die Entwicklung von Signalanalysealgorithmen und der anschließenden Datenanalyse von Messdaten. [16]

Parallel zu dieser Arbeit wurde an einer zweiten Masterarbeit, die sich mit der Umsetzung der Hardware beziehungsweise der Messkette beschäftigt [17], gearbeitet. Als Schnittstelle zwischen Messung und Analyse und damit zwischen den Arbeiten dienen die Messdaten. Außerdem sollen die Algorithmen als Basis für die weitere Forschungsarbeit dienen und auch bei Brennstoffzellen zum Einsatz kommen.

Die Arbeit umfasst die Literaturrecherche bezüglich Elektrochemische Impedanzspektroskopie mit besonderem Augenmerk auf die Anwendung an Elektrolyse Zellen und Stacks. Die Einarbeitung in die Programmiersprache JULIA [18] und die Versionskontrolle mittels GITLAB.[19] Das Erstellen der notwendigen Algorithmen für die Datenanalyse sowie deren Dokumentation. Das Aufbereiten und zur Verfügung stellen der Messergebnisse in Form von Daten und Grafiken. Die Analyse und Interpretation der Ergebnisse hinsichtlich Leistungsfähigkeit und Degradation aufgrund verschiedener Parameter.

15 Vgl. Europäische Kommision, CORDIS Forschungsergebnisse der EU 2023.
16 Vgl. HyCentA Research GmbH 2022.
17 Vgl. Joshua Eder 2023.
18 Vgl. Julia Lab at MIT 2023.
19 Vgl. GitLab B.V 2023.

3 GRUNDLAGEN DER WASSERSTOFFTECHNIK

Wasserstoff ist nicht nur eines der wichtigsten Elemente in der chemischen und petrochemischen Industrie, sondern auch das am häufigsten vorkommende Element im Universum. Wasserstoff tritt auf der Erde jedoch nur in gebundener Form als Wasser, in Kohlenwasserstoffen oder in Mineralien auf [20], was seiner Reaktionsfreudigkeit geschuldet ist. Um die Grundlagen und das Zusammenhängen der Wasserstofftechnik und im Besonderen die Elektrolyse zu verstehen, werden in diesem Kapitel die Stoffeigenschaften sowie die technischen Rahmenbedingungen für die Herstellung behandelt.

3.1 Stoffeigenschaften von Wasserstoff

Wasserstoff tritt in seiner atomaren Form in drei Isotopen auf, welche in Abbildung 3.1 dargestellt sind. Das mit 99,9 % am häufigsten vorkommende Wasserstoffisotop ist Protium, welches aus einem Proton im Kern und einem Elektron in der Hülle besteht. Schwerer Wasserstoff wird als Deuterium bezeichnet und besitzt, zusätzlich zum Proton, im Kern ein Neutron und tritt nur mit einer Häufigkeit von 0,015 % auf. Tritium ist überschwerer Wasserstoff mit einem weiteren Neutron und hat damit circa 50 % mehr Masse als Deuterium, jedoch tritt dieses Isotop nur mit einer verschwindend geringen Häufigkeit auf. Außerdem ist Tritium instabil und radioaktiv [21]. Deuterium und Tritium kommen bei Fusionsreaktoren zum Einsatz, weshalb diese derzeit hauptsächlich in der Fusionsforschung eine Rolle spielen [22].

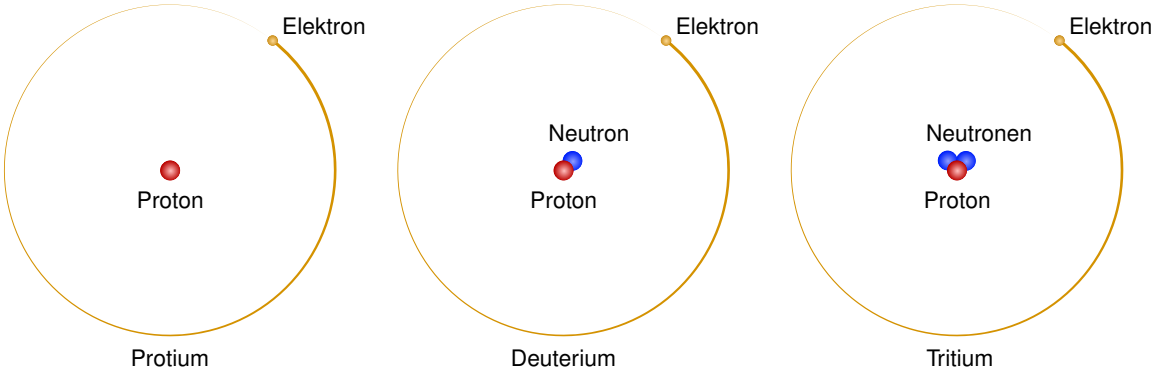

Abbildung 3.1: Isotope von Wasserstoff. Von links nach rechts Wasserstoff (Protium), schwerer Wasserstoff (Deuterium) und überschwerer Wasserstoff (Tritium). [23]

Ein Gasmolekül besteht immer aus zwei oder mehr Atomen und stellt das kleinste Teilchen dar, das noch die chemischen Eigenschaften des Gases besitzt, zu welchem es zugeordnet werden kann [24]. Die hohe Reaktionsfreudigkeit von atomarem Wasserstoff ist auf das fehlende Valenzelektron in der ersten Hülle zurückzuführen und dem damit einhergehenden Bestreben diese aufzufüllen. Deshalb verbindet sich das

[20] Vgl. Johannes Töpler, Jochen Lehmann (eds.) 2017, S. 4.
[21] Vgl. Manfred Klell, Helmut Eichlseder, Alexander Trattner; 2018, S. 56.
[22] Vgl. Garry McCracken, Peter Stott 2013, S. 33.
[23] Adaptiert von: Manfred Klell, Helmut Eichlseder, Alexander Trattner; 2018, S. 56.
[24] Vgl. Wolfgang Demtröder; 2016, S. 13.

Wasserstoffatom mit einem zweiten, über eine kovalente Bindung, zu molekularem Wasserstoff, wie Abbildung 3.2 veranschaulicht.

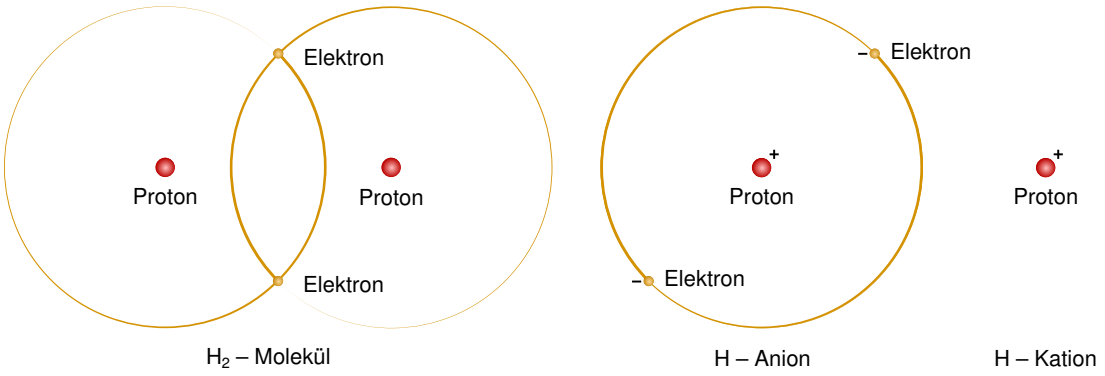

Abbildung 3.2: Wasserstoff-Molekül links sowie ein negativ geladenes Wasserstoff-Anion mittig und ein positiv geladenes Wasserstoff-Kation rechts.

Durch die Abgabe beziehungsweise Aufnahme von Elektronen verliert das Atom seinen neutralen Ladungszustand nach außen hin. Durch die Aufnahme eines weiteren Elektrons ist der Ladungszustand nach außen hin negativ, was als Anion bezeichnet wird. Durch die Abgabe des Elektrons ist nur mehr der positiv geladene Kern vorhanden, weshalb das Potenzial nach außen positiv ist und als Kation bezeichnet wird. Diese geladenen Teilchen sind für die Funktion der Elektrolyse entscheidend, da diese zugeführten Elektronen einen Stromkreis bilden.

3.2 Grundlagen der Elektrolyse

Wasser wird durch Anlegen einer Gleichspannung in einem elektrochemischen Prozess in seine Bestandteile Wasserstoff und Sauerstoff aufgespalten. Der klassische Elektrolyseversuch[25] mit zwei Elektroden in einem Wasserbad ist in Abbildung 3.3 schematisch dargestellt. Fließt der Strom zwischen zwei Elektroden, die als Anode und Kathode bezeichnet werden, wird das Wassermolekül H_2O in seine Bestandteile aufgespaltet. Dabei wird elektrische in chemische Energie umgewandelt. Die Reduktion und damit die Aufnahme von Elektronen sowie die Abgabe von Wasserstoff findet gemäß der Reaktionsgleichung

$$2\,H_2O + 2\,e^- \longrightarrow H_2 + 2\,OH^-$$

an der Kathode statt. Durch den Elektronenüberschuss, welchen die außen liegende Spannungsquelle erzeugt, wird diese zum negativen Pol. Der Elektrolyt, in diesem Fall Wasser, übernimmt den Transport der Hydroxidionen OH^-. Es gibt auch Verfahren mit Wasserstoffkationen H^+ im Elektrolyten, die den Ladungstransport übernehmen. Die Ionen fließen aufgrund ihrer negativen Ladung in Richtung des positiven Pols, der Anode. Hier findet die Oxidation und damit die Abgabe von Elektronen sowie des Sauerstoffs, nach der Reaktionsgleichung

$$4\,OH^- \longrightarrow 2\,H_2O + O_2 + 4\,e^-$$

statt. Es bildet sich Wasser und Sauerstoff, wobei das Wasser den Reaktionszyklus an der Kathode erneut durchläuft. Die Elektronen fließen zurück zur Kathode, wodurch sich die Anode ihre positive Polarität behält.[26]

[25] Vgl. August Wilhelm von Hofmann 1866, S. 55.
[26] Vgl. Alfredo Ursua, Luis M. Gandia, Pablo Sanchis; 2012, S. 412.

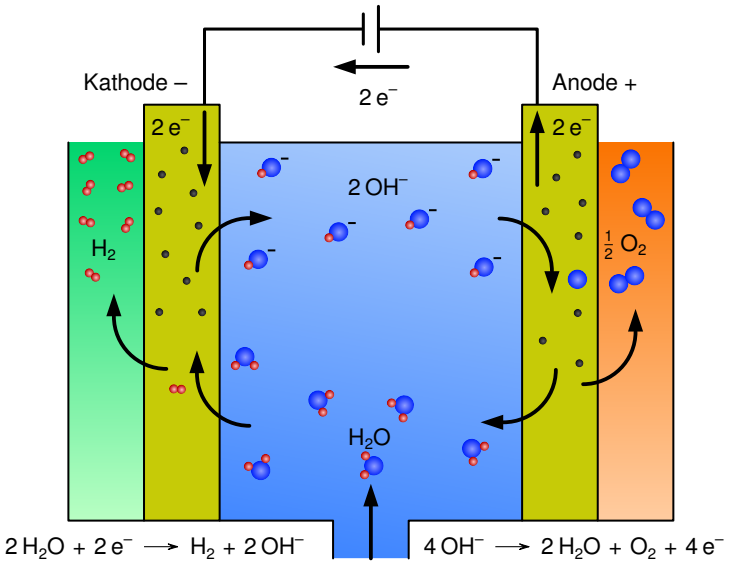

Abbildung 3.3: Prinzip des basischen Elektrolysevorgangs bei dem sich an der Kathode Wasserstoff und an der Anode Sauerstoff bildet. [27]

Die Gesamtreaktion einer Elektrolysezelle ergibt sich somit wie folgt

$$2\,H_2O(l) \longrightarrow 2\,H_2(g) + O_2(g),$$

wobei (l) für die flüssige und (g) für gasförmige Phase steht.

3.3 Industrielle Verfahren zur Herstellung von Wasserstoff

Wie bereits erwähnt, kann Wasserstoff mit unterschiedlichen Verfahren und Prozessen gewonnen werden.

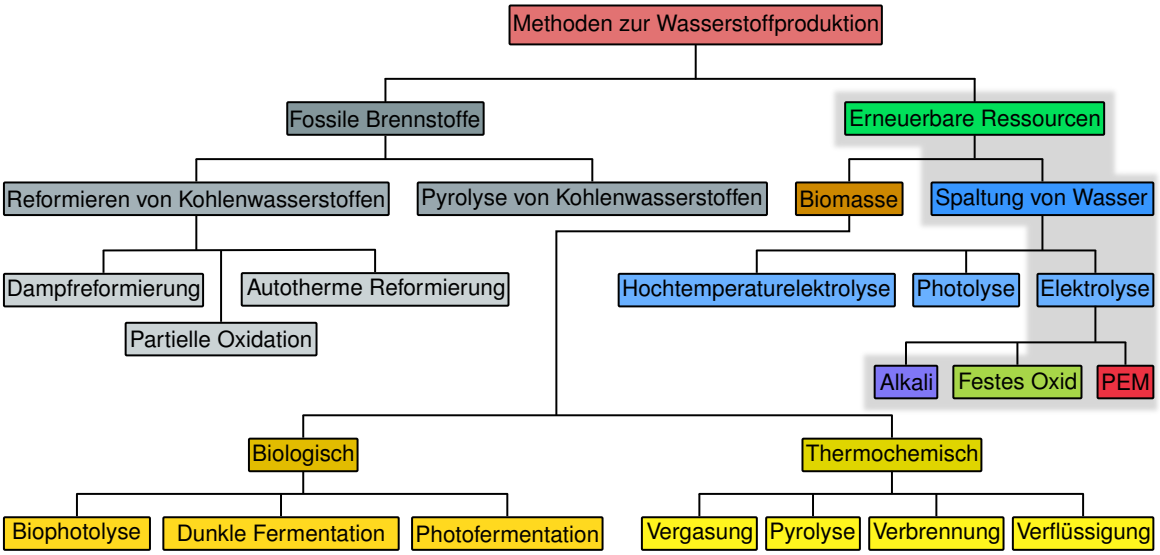

Abbildung 3.4: Überblick über die Verfahren zur Wasserstoffproduktion. [28]

[27] Adaptiert von: Manfred Klell, Helmut Eichlseder, Alexander Trattner; 2018, S. 74.

Der Großteil des produzierten Wasserstoffs wird in der Produktion von Dünger, dem Raffinieren von Öl, in der Petrochemie, der Halbleiterherstellung und in der chemischen Industrie verbraucht. Die Vielzahl der Herstellungsmöglichkeiten wird in Abbildung 3.4 aufgezeigt. Dabei wird zwischen der Erzeugung aus fossilen Brennstoffen und erneuerbaren Ressourcen unterschieden.

Ein Auszug dieser fossil gewonnenen Gase und deren atomare Zusammensetzung ist in Abbildung 3.5 aufgelistet. Diese enthalten zumeist Kohlenwasserstoffe, deren Verbindungen aufgespaltet werden. Der größte Teil der Wasserstoffproduktion, mit über 90 %, entfällt dabei auf die Dampfreformierung von Methan und damit auf fossile Energieträger. Dies hat den Nachteil, dass CO_2 emittiert und ein schlechter Reinheitsgrad des produzierten Produktgases, im Vergleich zur Elektrolyse, erzielt wird.

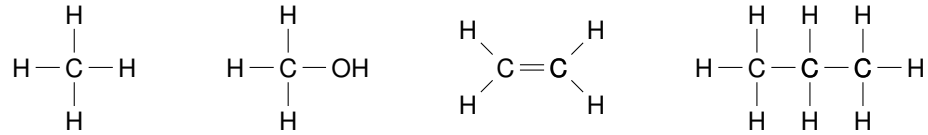

Abbildung 3.5: Kohlenwasserstoffe von links nach rechts: Methan CH_4, Methanol CH_3OH, Ethen C_2H_4 und Propan C_3H_8.

Mit Hilfe von grünem Wasserstoff in Kombination mit Kohlenstoff, der aus CO_2 gewonnen wird, kann wieder synthetischer Kraftstoff und Gas hergestellt werden. Hierbei sind die Emissionen bei der Verbrennung zumindest klimaneutral sind. Somit ist es möglich auch zukünftig Kohlenwasserstoffe für industrielle Anwendungen zu verwenden, wenn diese klimaneutral erzeugt werden.

Auch aus Biomasse kann Wasserstoff gewonnen werden, die derzeitigen Produktionsmengen sind aber noch nicht so groß. Der zukünftig wichtigste Zweig ist die Spaltung von Wasser mittels Elektrolyse, da hier ein hoher Reinheitsgrad erreicht wird und keine umweltschädlichen Nebenprodukte anfallen. In dieser Kategorie haben die alkalische und PEM (Proton Exchange Membran) Elektrolyse derzeit die größte Bedeutung, weshalb auf diese Verfahren in den folgenden Unterkapiteln kurz eingegangen wird.

3.3.1 Alkalische Elektrolyse

Die alkalische Elektrolysezelle ist seit gut 100 Jahren bekannt. Der schematische Aufbau und die Reaktionsgleichungen an Anode und Kathode sind in Abbildung 3.6 illustriert.

Das Wasser wird an der Kathodenseite zugeführt und auf der Anodenseite zusätzlich gebildet. Die beiden Tanks sind mit wässriger 20 wt% bis 40 wt%iger Kalilauge (KOH) gefüllt, die Hydroxidionen OH^- leitet. Ein Diaphragma fungiert als Separator, der den Anoden- vom Kathodenbereich trennt und somit auch die beiden Prozessgase. Die Stromzufuhr erfolgt über die Endplatten die wiederum leitend mit den Elektroden verbunden sind. Die beiden Tanks dienen zusätzlich als Gas-Flüssig-Separator, in denen sich im oberen Bereich die Prozessgase ansammeln und abgeführt werden können.

Der große Vorteil der alkalischen Elektrolyse ist, dass keine seltenen Erden gebraucht werden, da die Katalysatoren aus Nickel bestehen. Nachteil des flüssigen und korrosiven Elektrolyten ist, dass im Nachgang eine aufwändige Gasreinigung notwendig ist. Stand der Technik sind mittlerweile Anlagen im Megawattbereich bei Betriebstemperaturen von 50 °C bis 80 °C. [29] [30]

[28] Adaptiert von: Shiva Kumar und Himabindu; 2019, S. 443.
[29] Vgl. Peter Kurzweil, Otto K. Dietlmeier (auth.) 2015, S. 432 – 433.
[30] Vgl. Manfred Klell, Helmut Eichlseder, Alexander Trattner; 2018, S. 81 – 83.

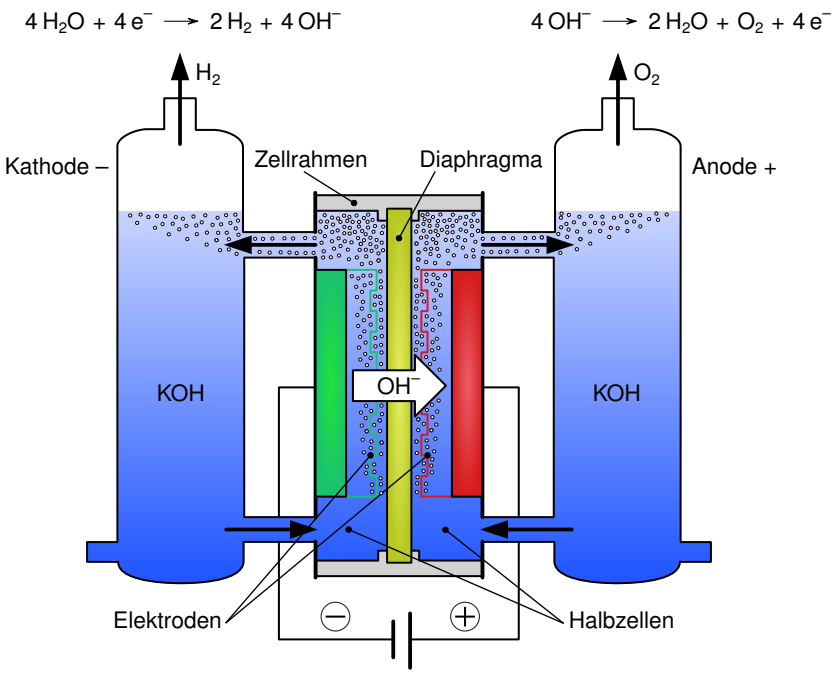

Abbildung 3.6: Prinzipieller Aufbau einer alkalischen Elektrolysezelle. [31]

3.3.2 Wasser Elektrolyse mittels Protonen-Austausch-Membran (PEM)

Die PEM-Elektrolysezelle, die auf dem Konzept eines festen Polymerelektrolyten beruht, wurde ursprünglich für den Einsatz in der Schwerelosigkeit auf einer Raumstation entwickelt, um dort Sauerstoff zu generieren. Den hohen Anforderungen an die Reinheit des produzierten Wasserstoffes sowie der kommerziellen Nachfrage ist es zu verdanken, dass diese kommerziell genutzt wird. [32]

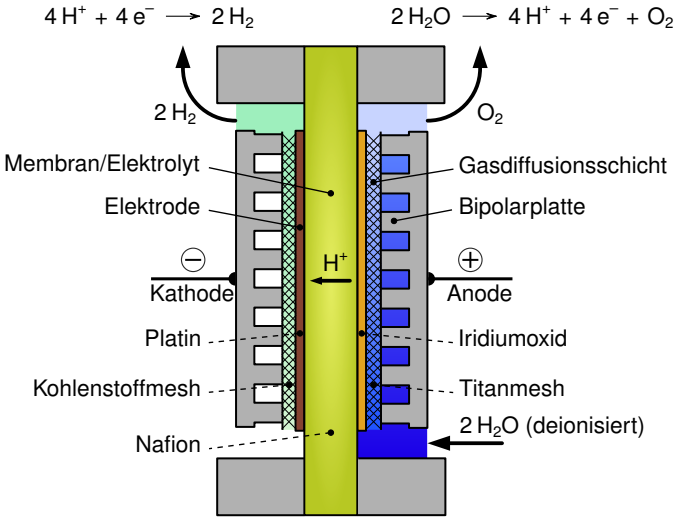

Abbildung 3.7: Schematischer Aufbau einer PEM-Elektrolysezelle mit den wichtigsten Materialien. [33]

Die feste und H^+-leitende Membran besteht aus sulfoniertem Tetrafluorethylen-Polymer und wird vom Hersteller Dupon unter der Bezeichnung Nafion® geführt. Sie vereint die Funktionen des Diaphragmas und des

[31] Adaptiert von: Tom Smolinka, Martin Günther, Jürgen Garche; 2011, S. 10.
[32] Vgl. Manfred Klell, Helmut Eichlseder, Alexander Trattner; 2018, S. 83 – 84.
[33] Adaptiert von: Tom Smolinka, Martin Günther, Jürgen Garche; 2011, S. 13.

protonenleitenden Elektrolyten, womit die Zelle nur mit deionisiertem Wasser versorgt werden muss, wie Abbildung 3.7 verdeutlicht. Direkt mit der Membran sind die beiden Elektroden verbunden, die an der Anode mit Iridiumdioxid (IrO_2) beschichtet ist und an der Kathode mit Platin. Darauf folgt die Gasdiffusionsschicht, die auf der Anodenseite aus einem Titanmesh und auf der Kathodenseite aus einem Kohlenstoffmesh besteht. Die Gase, die sich an den Elektroden bilden, können durch das poröse Material innerhalb der Zelle aufsteigen und so separat abgeführt werden. Die Materialien sind elektrisch leitend, da durch sie die elektrische Verbindung zwischen den Bipolarplatten und den Elektroden hergestellt wird. Außerdem sind in den Bipolarplatten die Strömungskanäle für den Wassertransport integriert.

Im Gegensatz zur alkalischen Elektrolyse, muss das zugeführte Wasser von hoher Reinheit sein. Schon geringe Konzentrationen von Schadstoffen können die Katalysatoren (Iridiumdioxid und Platin) sowie die Membran kontaminieren. Aufgrund des Membranmaterials ist die Betriebstemperatur auf $80\,^{\circ}C$ beschränkt. Es werden Stromdichten von bis zu $10\,A\,cm^{-2}$ und Anschlussleistungen für Stacks von über $2\,MW$ erreicht [34]. Außerdem eignet sich diese Zellenbauart besonders für den dynamischen Betrieb und ist auch für Teillasteinsätze gut geeignet. Besonders bei der volatilen Stromproduktion durch erneuerbare Energien ist dies ein großer Vorteil. Außerdem werden keine korrosiven Medien im Prozesswasser benötigt.

3.3.3 Stackaufbau

Als Stack, oder zu Deutsch auch als Stapel, wird der Verbund von mehreren in Reihe geschalteten Zellen bezeichnet. Dadurch ist es möglich, große Zellflächen und damit hohe elektrische Anschlussleistungen bei gleichzeitig kompakten Abmessungen zu erreichen. Stacks haben üblicherweise eine rechteckige oder runde Form. Die Spannungsversorgung erfolgt entweder über die beiden Kontaktplatten oder direkt über einen Stromsammler, wie Abbildung 3.8 illustriert.

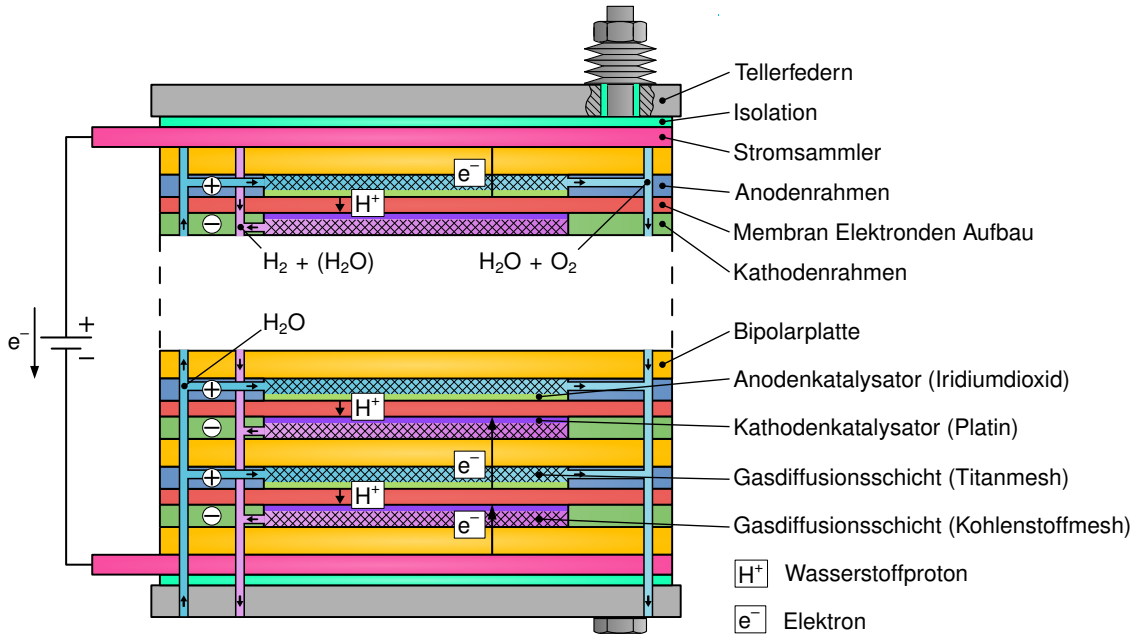

Abbildung 3.8: PEM-Stack mit Strömungskanälen und Flussrichtung der Wasserstoffionen sowie der Elektronen. [35]

Die Vorspannkraft wird von den Schrauben aufgebracht und über die Kontaktplatten gleichmäßig in den Stack eingeleitet. Zwischen der Kontaktplatte und den Schrauben sind Tellerfedern verbaut, wodurch

[34] Vgl. Agate Martin, Patrick Trinke, Boris Bensmann, Richard Hanke-Rauschenbach 2022.
[35] Adaptiert von: Haas Christoph; 2018, S. 44.

die Materialdehnung aufgrund von Temperaturschwankungen ausgeglichen wird. Außerdem müssen die Schrauben von den Kontaktplatten sowie dem Stack elektrische getrennt werden, um keinen Kurzschluss zu verursachen.

Der Aufbau des restlichen Stapels erfolgt in einem wiederkehrenden Muster, je nach Anzahl der Zellen. Der Potenzialausgleich zwischen den einzelnen Zellen wird über die Bipolarplatten sichergestellt. Die Anoden- sowie der Kathodenrahmen dichten die Zellen nach außen hin ab und führen über Strömungskanäle den Wasserstoff sowie den Sauerstoff ab, weshalb die Rahmen die jeweilige Gasdiffusionsschicht sowie den Katalysator vollständig umschließen.

Der Wasserkreis führt an jeder Anode vorbei durch den Stack. Da sich an der Anode Sauerstoff bildet, tritt aus dem Stack ein Zweiphasengemisch aus flüssigem H_2O und gasförmigen O_2 aus. Außerdem lässt sich hier erkennen, dass sich über den Wasserkreis ein Kurzschluss zwischen den Zellen bilden würde, sofern nicht deionisiertes Wasser verwendet wird. Je niedriger die Reinheit des Wassers desto mehr parasitäre Ströme fließen zwischen den Zellen. Auch alle Strömungskanäle auf der Kathodenseite sind miteinander verbunden und führen an einem zentralen Punkt aus dem Stack heraus. Hier wird der produzierte Wasserstoff abgeschöpft, wobei sich im Gas auch Wasser befindet. Dies ist der hydrophilen Membran geschuldet, da diese erst leitfähig für Protonen wird, wenn sie feucht ist. Außerdem sind die Protonen von Wassermolekülen umgeben, wenn diese sich durch die Membran bewegen. Somit fallen im Wasserstoff nicht unerhebliche Anteile an Wasser an, weshalb die beiden Stoffe in einem nachgelagerten Prozess wieder separiert werden müssen.

Die Membran verläuft außerdem durch den gesamten Stackquerschnitt und ist für den Potenzialunterschied zwischen Anoden- und Kathodenseite verantwortlich, da diese nicht elektrisch leitend ist. Die Fließrichtung der negativ geladenen Elektronen und somit des Stromes ist entgegengesetzt zur Richtung der positiv geladenen Protonen.

3.3.4 Peripherie

Als Peripheriegeräte werden alle Anlagenteile bezeichnet, die zum Betrieb des Stacks benötigt werden. In Abbildung 3.9 ist der grundsätzliche Aufbau des hydraulischen Kreislaufes, sowie die Gasentnahmepunkte und die Prozesskühlung ersichtlich.

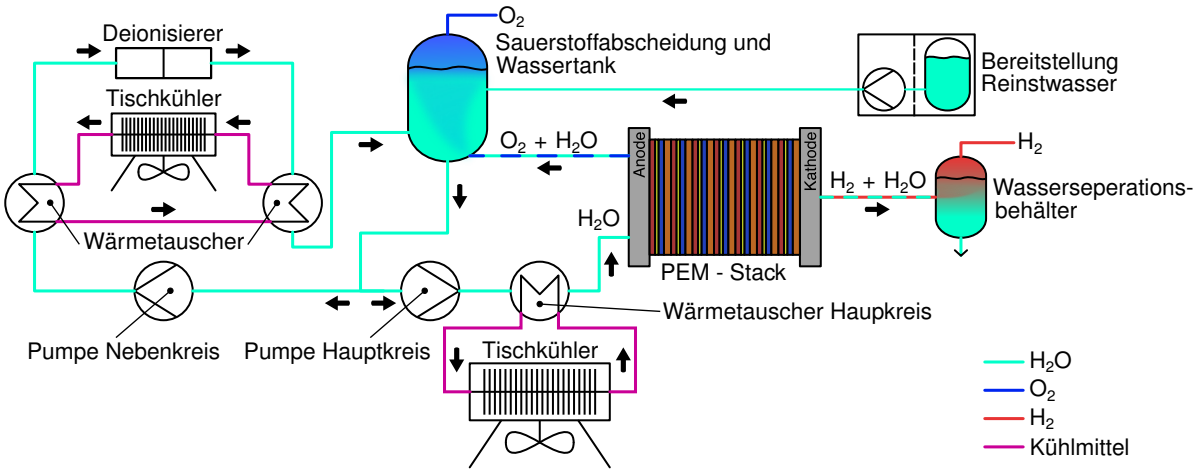

Abbildung 3.9: Anordnung der benötigten Peripheriegeräte, ohne die weitere Aufbereitung der entstehenden Prozessgase, sowie Leistungselektronik, Sensorik und Steuerung.

Für den Elektrolyseprozess wird Reinstwasser mit einer Leitfähigkeit von $< 0{,}1\,\mu S\,cm^{-1}$ benötigt, um einen Kurzschluss zwischen den Zellen zu verhindern. Deshalb wird bereits aufbereitetes Wasser zugeführt, das durch Verfahren, wie etwa Umkehrosmose oder durch Filtration hergestellt wird. Dieses wird dem System dann über den Wassertank zugeführt. Die Hauptpumpe lässt das Reinstwasser vom Wassertank aus durch den Stack zirkulieren. Dabei passiert es einen Kühler, der die entstehende Prozesswärme abführt. Dies ist notwendig, um die Prozesstemperatur konstant zu halten und die temperaturempfindliche Membran nicht zu schädigen. Außerdem kann hier auch Wärme eingebracht werden, um den Stack beim Anfahren des Systems auf die Betriebstemperatur zu bringen. Das aus dem Stack austretende Zweiphasengemisch aus O_2 und H_2O wird in den Tank eingeleitet und kann sich hier aufgrund des Dichteunterschiedes separieren. Der Sauerstoff wird an diesem Punkt abgeführt und durch Koaleszenzabscheider und Abkühlung weiter getrocknet.

Der zweite Wasserkreislauf dient zur kontinuierlichen Konditionierung des Systemwassers. Hierfür wird ein Teil des Prozesswassers über den Deionisiererkreis gepumpt und mittels Harze, die sich im Deionisierer befinden, gereinigt. Da die Maximaltemperatur dieser Harze unterhalb der Prozesstemperatur des Stacks liegt, muss das Wasser abgekühlt werden. Ein Teil der entnommenen Wärme wird, über einen zweiten Wärmetauscher und nach erfolgter Reinigung, wieder dem System zugeführt. Da die Temperaturdifferenz zwischen dem Kühlmittel im zweiten Wärmetauscher sowie dem Wasser nur gering ausfällt, muss der Großteil der Wärmeleistung über den Kühler entnommen werden. Da die Harze über die Zeit degradieren, müssen diese regelmäßig getauscht oder regeneriert werden.

Auf der Kathodenseite wird der Wasserstoff gewonnen. Wie in Absatz 3.3.3 beschrieben, kommt es auch hier zu einem Zweiphasengemisch von H_2 + H_2O. Deshalb ist dem Stack ein Absetzbehälter nachgeschaltet, wo sich der Großteil des Wassers absetzt. Auch hier wird das Gas anschließend noch weiter getrocknet, um dessen Reinheitsgrad zu erhöhen.

Die Sensoren sowie die Steuerung sind aus Abbildung 3.9 nicht ersichtlich. Zu überwachende Parameter sind die Temperaturen des Wassers sowie der Gase, die Leitfähigkeit des Prozesswassers, die Drücke im System sowie die Durchflussraten der Medien. So wird zum Beispiel die Temperatur des Stacks mit der Temperatur des Prozesswassers, bei konstantem Durchfluss durch den Stack, gesteuert. Eine weitere wichtige Komponente ist die Leistungselektronik und hier vor allem der Gleichrichter, welcher den Stack mit Energie versorgt und somit essentiell für das System ist.

4 GRUNDLAGEN DER SIGNALVERARBEITUNG

Als Signalverarbeitung wird die Signal- und Systemtheorie bezeichnet, die sich mit der Beschreibung, Analyse und Synthese von Systemen und dem Filtern von Informationen und Signalen befasst [36]. Signale können sich auf nicht-elektrische Größen beziehen, werden aber zum Großteil in Spannungs- und Stromsignale überführt, da diese leichter zu verarbeiten sind. Diese Wandlung der Größen erfolgt in der Sensorik, welche den Wert der jeweiligen physikalischen Größe bestimmt und den Signalausgang in Relation zu dieser setzt. Erst nach dieser Wandlung können die Signale effektiv verarbeitet werden. Aber bereits bei der Messung ist darauf zu achten, dass die Regeln der Systemtheorie, wie etwa eine ausreichende Abtastfrequenz, eingehalten werden. Im Falle der Impedanzspektroskopie erfolgt die Verarbeitung der Daten zumeist durch die Fast-Fourier-Transformation, welche als Ergebnis komplexe Zahlen liefert.

4.1 Komplexe Zahlenebene

Historisch gesehen wurden komplexe Zahlen eingeführt, um mit Polynomgleichungen wie

$$x^2 + 1 = 0, \tag{4.1}$$

die keine reelle Lösung enthalten, umzugehen [37]. Diese werden in der kartesischen Form als ein Set dargestellt

$$z = a + \mathrm{j}\, b \qquad \text{mit} \qquad \mathrm{j}^2 = -1 \tag{4.2}$$

welches aus Realteil $\mathrm{Re}(z) = a$ und Imaginärteil $\mathrm{Im}(z) = b$ besteht, wobei aber beide Teile über reelle Zahlen definiert sind. So wie reelle Zahlen eindimensional als Punkte auf einer Linie grafisch dargestellt werden können, sind komplexe Zahlen zweidimensional als Vektor der vom Ursprung auf den Punkt $(a,\, b)$ zeigt darstellbar. Deshalb lassen sich komplexe Zahlen auch als Zeiger r mit einem Winkel φ ausdrücken und mit den trigonometrischen Regeln gemäß

$$r = \sqrt{\mathrm{Re}(z)^2 + \mathrm{Im}(z)^2} \qquad \varphi = \arctan\left(\frac{\mathrm{Im}(z)}{\mathrm{Re}(z)}\right) \tag{4.3}$$

berechnen. Lässt man diesen Zeiger mit der Amplitude r und Frequenz ω mit einer Zeitdauer t um dessen Ursprung drehen, zeichnet dieser einen Kreis mit dem Radius r der über die reelle und imaginäre Achse aufgetragen wird und sich in der Exponentialform wie folgt anschreiben lässt

$$z = r\, \mathrm{e}^{\mathrm{j}\,\omega\, t}. \tag{4.4}$$

Zusätzlich kann dieser im dreidimensionalen Raum, durch Einführen der Zeit t, betrachtet werden, wodurch sich eine Spirale mit der Steigung t entlang der Zeitachse ausbildet, wie Abbildung 4.1 zeigt. Außerdem können komplexe Zahlen noch in der trigonometrischen Form (Eulersche Identität)

$$z = r\,(\cos\varphi + \mathrm{j}\, \sin\varphi) \tag{4.5}$$

[36] Vgl. Martin Meyer 2017, S. 1.
[37] Vgl. Eleanor Chu, Alan George 2000, Section 1.1.

angegeben werden. Durch die Umrechnung mit

$$\mathrm{Re}(z) = r\,\cos\varphi \qquad \mathrm{Im}(z) = r\,\sin\varphi \tag{4.6}$$

kann die Zahl wieder in die kartesische Form überführt werden und stellt sich grafisch als harmonische Kosinus- beziehungsweise Sinusschwingung des Real- und Imaginärteils dar.[38]

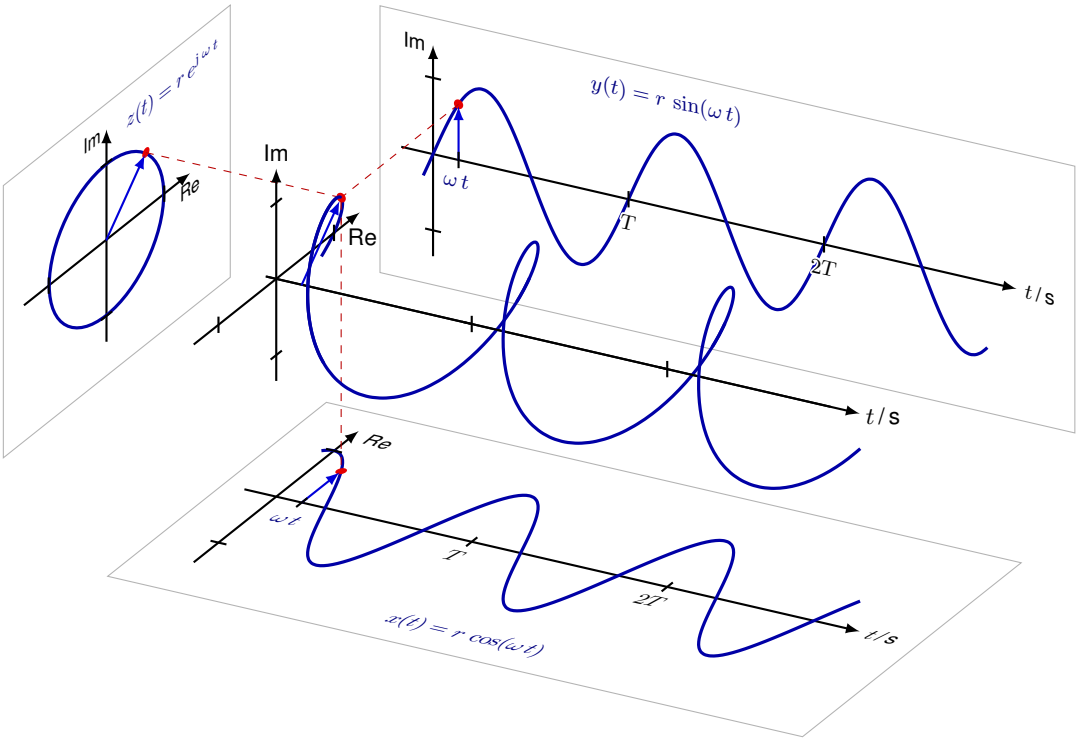

Abbildung 4.1: Komplexe Zahlenebene im zwei- und dreidimensionalen Raum.[39]

4.2 Nyquist-Shannon Abtasttheorem

Physikalische Größen sind immer zeitkontinuierlich und besitzen deshalb eine unendliche zeitliche Auflösung. Damit diese in eine zeit- und wertdiskrete Form überführt werden können, muss das Signal zu definierten Zeitpunkten abgegriffen werden. Die Dauer zwischen den Abgriffen (Sampling) wird als Abtastzeit bezeichnet und für jeden Tastpunkt ergeben sich zwei Zahlenwerte, die jeweils die Amplitude und den Abtastzeitpunkt beschreiben. Dabei postuliert das Nyquist-Shannon Abtasttheorem ein minimal erforderliche Abtastrate, die von der höchsten Frequenz im Signal bestimmt wird. Hierbei wird sichergestellt, dass eine Mindestanzahl an Punkten pro Periode in der Messung vorhanden ist. Wenn die Abtastrate gleich der Frequenz des Signals ist, reicht eine Linie um alle Datenpunkte zu schneiden, wie in Abbildung 4.2 grau dargestellt. Ist die Punkteanzahl pro Periode größer 1 und kleiner 2, kann eine Signal mit niedriger Frequenz alle Abtastpunkte schneiden, womit eine Wiederherstellung des ursprünglichen Inputs nicht garantiert werden kann, wie die gestrichelte Linie in Abbildung 4.2 verdeutlicht.[40]

[38] Vgl. Lothar Papula 2011, S. 654–655.
[39] Izaak Neutlings 2021, Adaptiert von:
[40] Vgl. Reinhard Lerch 2007, S. 311–313.

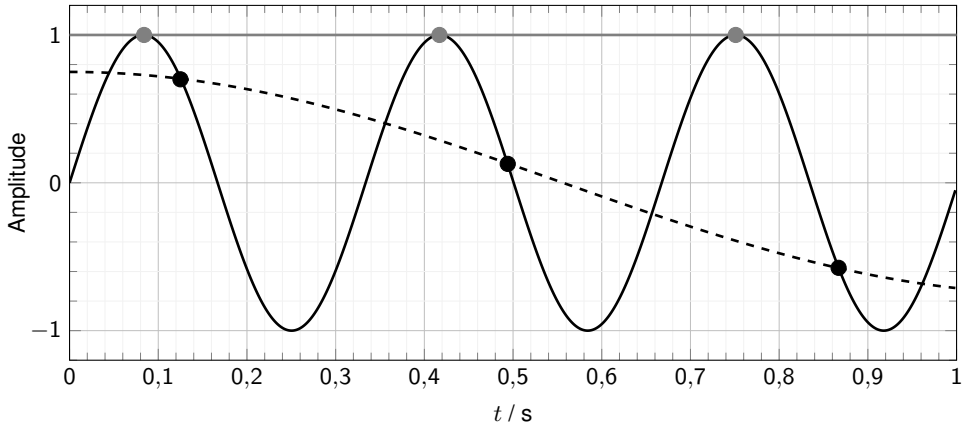

Abbildung 4.2: Aliasing-Effekte bei Unterschreitung der minimalen Abtastfrequenz.

Diese Fehler werden als Aliasing-Effekte bezeichnet. Deshalb muss die Abtastrate doppelt so groß wie die höchste Signalfrequenz, die reproduziert werden soll, sein. Da dies den Grenzfall darstellt wird in der Technik ein größerer Wert gewählt, womit die Abtastrate wie folgt definiert ist

$$f_{abtast} > 2{,}2 \cdot f_{signal}. \tag{4.7}$$

Selbes gilt für Frequenzen, die größer als die halbe Abtastfrequenz sind. Hier können hochfrequente Signale durch alle Abtastpunkte verlaufen, sofern die Signalfrequenz ein Vielfaches der Abtastfrequenz ist, wie Abbildung 4.3 illustriert. Dies ist bei der Auswertung der Messpunkte zu beachten, das heißt die maximale Frequenz muss auf die halbe Abtastrate beschränkt werden, da ansonsten keine eindeutige Zuordnung möglich ist.

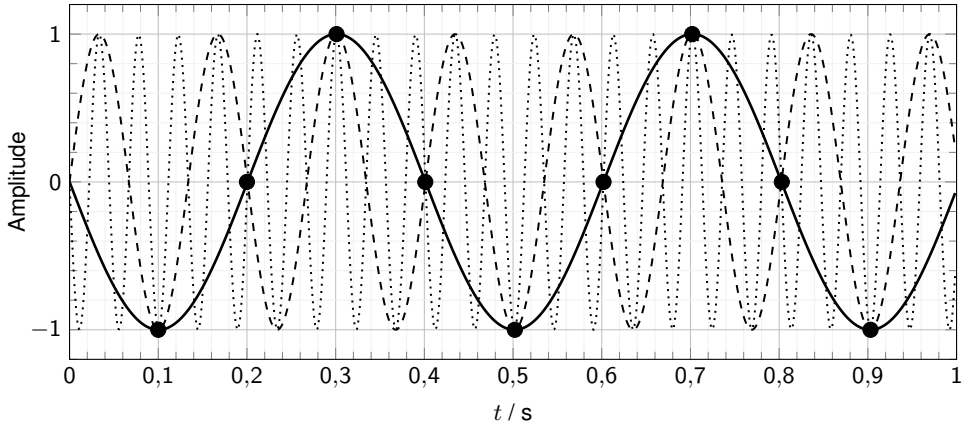

Abbildung 4.3: Auszug aus drei Signalfrequenzen die mit den selben Abtastpunkten reproduziert werden können. Neben der Frequenz die der halben Abtastrate entspricht (durchgezogene Linie), kann eine beliebige Anzahl an Signalen mit dem vielfachen dieser Frequenz durch die Messpunkte verlaufen (gestrichelte Linien).

4.3 Fourier-Transformationen

Signale können entweder im Zeit- oder im Frequenzbereich untersucht werden. Im Zeitbereich geht es um den Momentanwert des Signals zu einem bestimmten Zeitpunkt. Im Frequenzbereich werden die im Signal vorkommenden Frequenzen sowie deren Stärke betrachtet. Beide Darstellungsarten sind für harmonische

Schwingungen gleichwertig, da sie alle Informationen enthalten und in die jeweils andere überführt werden können. Um das Zeitsignal in seine Frequenzen zu zerlegen, wird die Fourier-Transformation angewandt, welche zugleich einen der wichtigsten signaltechnischen Prozesse darstellt [41]. Der Vollständigkeit halber sei erwähnt, dass es noch den vier Parameter Fit und den Goertzel Algorithmus gibt, um ein Signal vom Zeitbereich in den Frequenzbereich zu transferieren. Die Fourier-Transformation ist durch Bildung ihrer Inversen invertierbar, was besonders in der digitalen Telekommunikation von Bedeutung ist. So muss nur die Information der Frequenzen und nicht das gesamte Signal übertragen werden. Das Prinzip der Transformation ist in Abbildung 4.4 dargestellt und veranschaulicht grafisch den Zusammenhang zwischen dem Zeitbereich und dem Frequenzbereich.

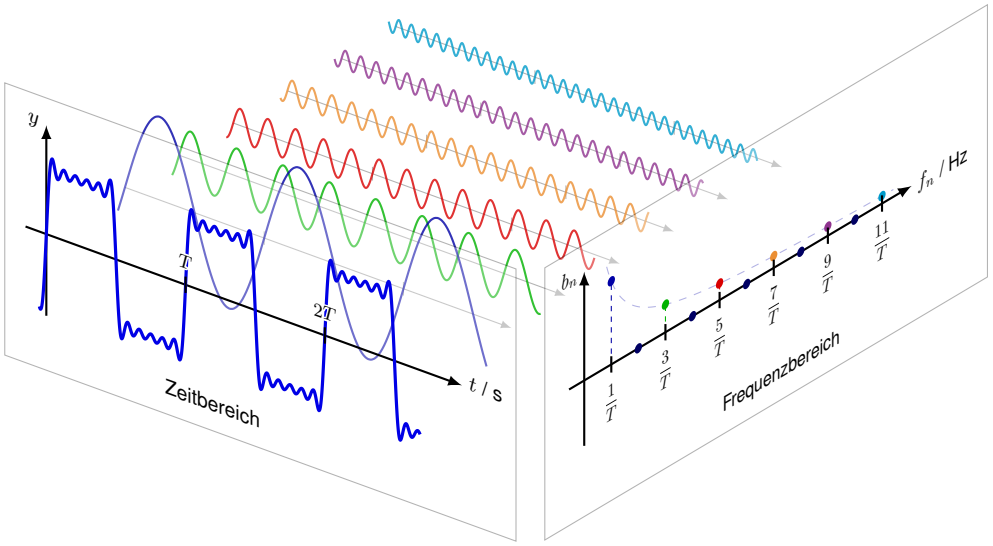

Abbildung 4.4: Bildliche Darstellung der Überführung von Signalen vom Zeitbereich in den Frequenzbereich mittels Fast Fourier Transformation. [42]

Grundsätzlich ist die Fast-Fourier-Transformation (FFT) einer der wichtigsten Algorithmen in der modernen computergestützten Signalverarbeitung. Das liegt daran, dass diese in der Bildverarbeitung, Komprimierung von Audiofiles, Kommunikationsnetzwerken, numerischen Physik und der Datenanalyse tief in der digitalen Welt verankert ist [43].

4.3.1 Fourierreihe

Die Basis für den FFT-Algorithmus bildet die analytische Fourier-Reihe, welche sich aus unendlich vielen Sinus- und Kosinusfunktionen mit steigender Frequenz zusammensetzt. Durch die Summation der Winkelfunktionen bei verschiedenen Frequenzen ist es möglich eine beliebige periodische Funktion $f(t)$ anzunähern. Geht die Entwicklung der Reihe gegen unendlich, entspricht diese genau $f(t)$. Somit kann jede noch so komplexe periodische Funktion durch eine Summe einfacher Funktionen ersetzt werden, deren Berechnung leichter fällt und nur iterativ wiederholt werden muss. In der Praxis wird die Anzahl der Iterationen k entweder vorgegeben oder über ein Kriterium, wie etwa dem mittleren quadratischen Fehler, beschränkt.

41 Ulrich Karrenberg 2016, Vgl.
42 Adaptiert von: Izaak Neutlings 2021.
43 Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 54.

Die mathematische Definition der Fourierreihe für $f(t)$ mit einer Periodendauer von $2\,\pi$ lautet

$$f(t) \approx \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(kt) + b_k \sin(kt)\right). \tag{4.8}$$

Die beiden Terme a_k und b_k werden dabei als Fourierkoeffizienten bezeichnet, welche bestimmen wie hoch der Amplitudenausschlag in der jeweiligen Frequenz ist, womit diese unterschiedlich stark in die Summe eingehen. Beim Term vor der Summe handelt es sich um einen Versatz des Signales, wie etwa einen Gleichstromanteil. Angenommen das Ausgangssignal $f(x)$ deckt einen Bereich von $-\pi$ bis π ab, dann werden die Koeffizienten wie folgt definiert: [44]

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos(kx)\,\mathrm{d}t \qquad \text{für} \qquad k \geq 0, \tag{4.9}$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin(kx)\,\mathrm{d}t \qquad \text{für} \qquad k \geq 1. \tag{4.10}$$

Wie beliebige periodische Funktionen, in diesem Fall eine Rechteck- sowie eine Sägezahnfunktion, mit einer finiten Anzahl an Durchläufen angenähert werden, ist in Abbildung 4.5 illustriert. Dabei gilt, die Anzahl der Iterationen verhält sich reziprok zum mittleren Fehler der Näherung. Außerdem können diese Art von Funktionen nie ganz angenähert werden, weil es sich um unstetige Funktionen mit Sprüngen handelt. [45]

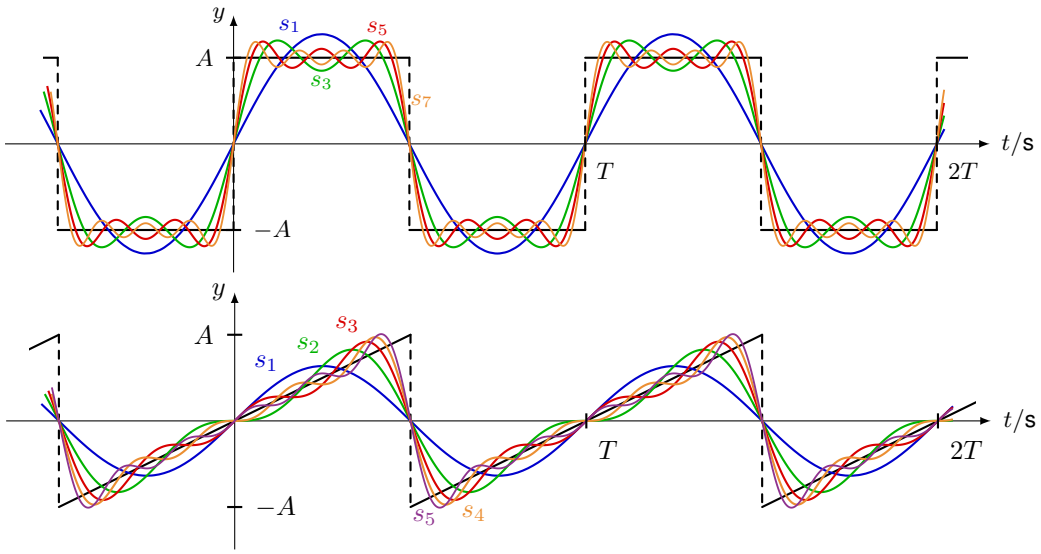

Abbildung 4.5: Rechteck- und Sägezahnfunktion angenähert mittels Fourierreihe. [46]

4.3.2 Fourier-Transformation

Die Fourierreihe ist als Funktion, die sich periodisch wiederholt, definiert, weshalb sich diese auch außerhalb der definierten Domäne fortlaufend repliziert. Das Integral der Fourier-Transformation stellt die Grenze der Fourierreihe dar, wenn die Länge des Betrachtungsbereichs gegen unendlich geht. Deshalb ist diese Funktion, ohne sich zu wiederholen, von $(-\infty, \infty)$ definiert, wie Abbildung 4.6 zeigt.

44 Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 56.
45 Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 57–58.
46 Adaptiert von: Izaak Neutlings 2021.

Wenn man die Fourierreihe im Bereich von $(-L, L)$ betrachtet und dann $L \to \infty$ laufen lässt, stellt sich diese wie folgt dar

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[a_k \cos\left(\frac{k\pi x}{L}\right) + b_k \sin\left(\frac{k\pi x}{L}\right) \right] = \sum_{k=-\infty}^{\infty} c_k \, \mathrm{e}^{\frac{\mathrm{i}k\pi x}{L}}, \tag{4.11}$$

wobei sich der Koeffizient c_k wie folgt ergibt

$$c_k = \frac{1}{2L} \int_{-L}^{L} f(t) \, \mathrm{e}^{\frac{\mathrm{i}k\pi x}{L}} \, \mathrm{d}x. \tag{4.12}$$

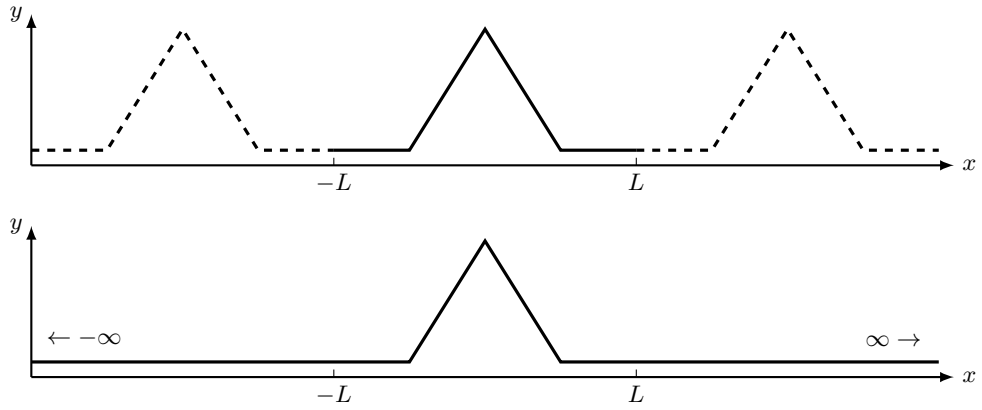

Abbildung 4.6: Oben die periodische Funktion der Fourierreihe und unten die aperiodische Funktion für die Fourier Transformation.

Die Funktion wird durch eine Summe aus Sinus- und Kosinusfunktion beschrieben, wobei nur diskrete Frequenzen verwendet werden, da $\omega_k = k\pi/L$ und k nur einen ganzzahligen Wert annehmen kann. Verschiebt man nun die Grenze $L \to \infty$, erhält man ein kontinuierliches Frequenzband. Dabei konvergiert die Schrittweite $\Delta\omega = \pi/L$ gegen Null

$$f(x) = \lim_{\Delta\omega \to 0} \sum_{k=-\infty}^{\infty} \frac{\Delta\omega}{2\pi} \int_{-\pi/\Delta\omega}^{\pi/\Delta\omega} f(\xi) \, \mathrm{e}^{-\mathrm{i}k\Delta\omega\xi} \, \mathrm{d}\xi \; \mathrm{e}^{\mathrm{i}k\Delta\omega x}, \tag{4.13}$$

womit man die Fourier-Transformation $\hat{f}(\omega) \triangleq \mathcal{F}(f(x))$ erhält. Darüber hinaus wird die Summation mit einer Gewichtung von $\Delta\omega$ zu einem Riemannschen Integral, was zu folgendem Ergebnis führt

$$f(x) = \mathcal{F}^{-1}\left(\hat{f}(\omega)\right) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) \, \mathrm{e}^{\mathrm{i}\omega x} \, \mathrm{d}\omega, \tag{4.14}$$

$$\hat{f}(\omega) = \mathcal{F}(f(x)) = \int_{-\infty}^{\infty} f(x) \, \mathrm{e}^{-\mathrm{i}\omega x} \, \mathrm{d}x. \tag{4.15}$$

Die beiden Integrale werden auch das Fourier-Transform Paar genannt. [47]

[47] Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 62–63.

4.3.3 Diskrete Fourier Transformation

Bis zu diesem Punkt wurde die Fourierreihe und die Transformation nur für kontinuierliche Funktionen $f(x)$ betrachtet. Wenn mit Messdaten gearbeitet wird, handelt es sich jedoch um diskrete Datensätze in Abhängigkeit der Abtastrate, wie schon in Abschnitt 4.2 beschrieben.

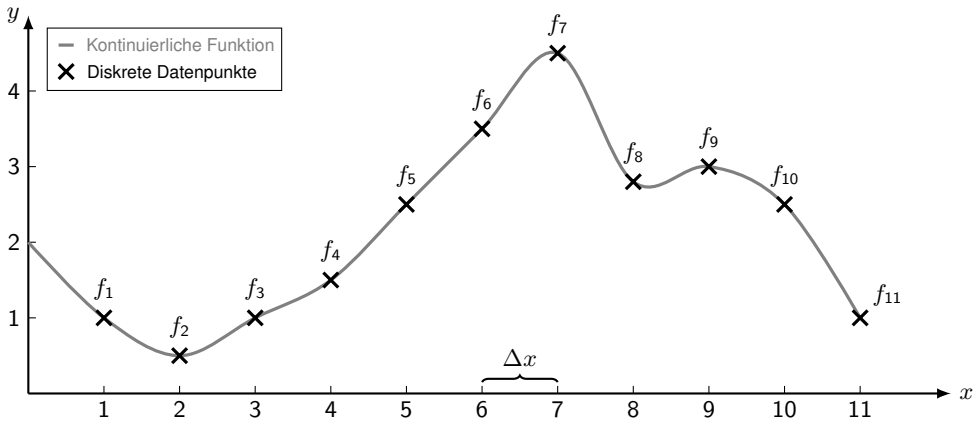

Abbildung 4.7: Funktion in kontinuierlicher und diskreter Form.

Deshalb ist die diskrete Fourier-Transformation (DFT) die diskretisierte Version einer Fourierreihe mit einem Abstand von Δx zwischen den Datenpunkten, wie in Abbildung 4.7 verbildlicht.

Die Formulierung der diskreten Fourier-Transformation, die nun von der Anzahl der Datenpunkte abhängt, lautet wie folgt

$$\hat{f}_k = \sum_{\mathrm{i}=0}^{n-1} f_\mathrm{i}\, \mathrm{e}^{\,-\mathrm{i}2\pi k/n}, \tag{4.16}$$

und deren Inverse

$$f_k = \frac{1}{n} \sum_{\mathrm{i}=0}^{n-1} \hat{f}_\mathrm{i}\, \mathrm{e}^{\,\mathrm{i}2\pi k/n}. \tag{4.17}$$

Da die DFT ein linearer Operator ist, werden die Datenpunkte vom Zeitbereich f in den Frequenzbereich $\hat{f}$ gehoben

$$\{f_1, f_2, \ldots, f_n\} \overset{\text{DFT}}{\Longrightarrow} \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_n\}. \tag{4.18}$$

Für eine Anzahl von n Punkten, werden die Daten mit der Sinus- und Kosinusfunktion und der zugrundeliegenden Frequenz $\omega_n = \mathrm{e}^{-j2\pi/n}$ repräsentiert. Diese kann als Matrixmultiplikation durchgeführt werden

$$\begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix}. \tag{4.19}$$

Der Ergebnisvektor $\hat{f}$ beinhaltet die Fourierkoeffizienten vom Eingangsvektor f. Die DFT-Matrix ist komplex, womit der Ergebnisvektor eine Amplitude und eine Phasenwinkel enthält, was besonders bei der physikalischen Interpretation von Bedeutung ist. [48]

[48] Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 65–66.

4.3.4 Schnelle Fourier Transformation

Die diskrete Fourier-Transformation ist sehr nützlich für die computergestützte numerische Näherung. Sie skaliert jedoch schlecht mit der Anzahl an Datenpunkten, was besonders bei großen Datensätzen einen erheblichen Nachteil darstellt. Dies macht die Notation deutlich, da eine Multiplikation der $n \times n$ Matrix nötig ist, was den Rechenaufwand mit der Punktezahl quadratisch anwachsen lässt. Deshalb wurde die schnelle Fourier Transformation (FFT) entwickelt, die nur mit $n \cdot \log(n)$ skaliert. Wird n sehr groß, wächst die Komponente $\log(n)$ nur langsam an, wodurch die Funktion nahezu linear skaliert. Der Algorithmus basiert auf einer fraktalen Symmetrie der Fourier-Transformation, welche es erlaubt eine $n = 2^p$ dimensionale DFT durch DFTs einer kleineren Dimension zu lösen. Die FFT stellt einen der wichtigsten Algorithmen für die moderne Kommunikation und Datenübertragung dar. So hat diese die Komprimierung von Audio und Bilddateien in MP3 und JPEG sowie Videostreaming, Satellitenkommunikation und das Mobilfunknetz, um nur einige zu nennen, erst möglich gemacht. [49]

4.3.5 Programmschritte einer FFT-Analyse

Um die Amplitudenhöhen und die dazugehörigen Frequenzen, aus denen sich ein Signal zusammensetzt, bestimmten zu können, sind drei Schritte notwendig, wie Abbildung 4.8 zeigt.

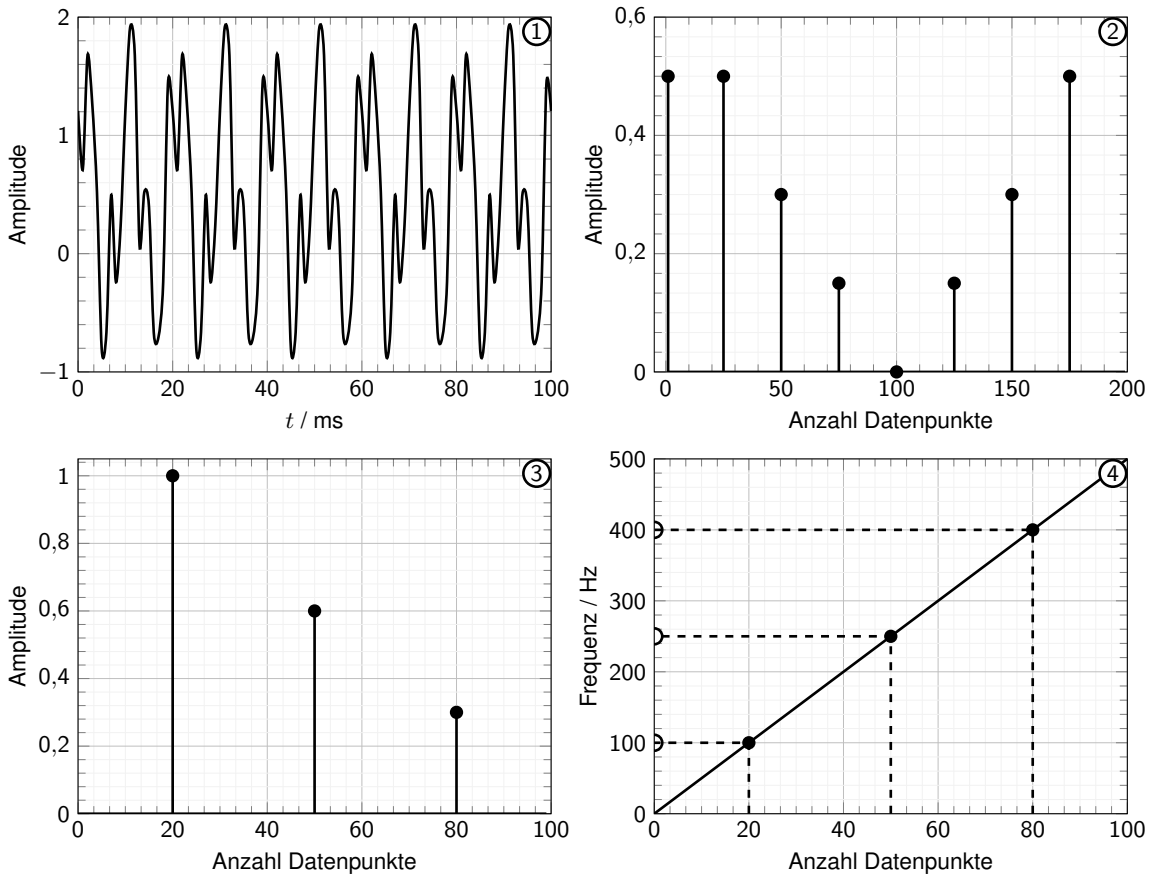

Abbildung 4.8: Die nötigen Programmschritte einer FFT-Analyse um aus dem Messsignal das Frequenzspektrum zu generieren. 1. Rohdaten, 2. gesamtes Amplitudenspektrum, 3. gespiegeltes Amplitudenspektrum, 4. Frequenzvektor.

[49] Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 65, 68.

Der erste Abschnitt zeigt ein Signal, welches sich aus drei überlagerten Sinussignalen zusammensetzt. Dieses Signal stellt die Rohdaten und somit den Input für die FFT dar. Im zweiten Bild ist der Amplitudenplot der FFT-Daten ersichtlich, wobei diese in halber Amplitudenhöhe und um die Mitte der Datenpunkte gespiegelt sind. Hierfür wird aus allen Datenpunkten der Betrag des Imaginär- und Realteiles gebildet, welche die FFT für jeden Datenpunkt ausgibt und mit der Punkteanzahl des Datensatzes normiert. Das Normieren ist notwendig, da die Anzahl der Datenpunkte, die aufsummiert werden, variiert, dies aber keinen Einfluss auf den Amplitudenwert haben darf. Zusätzlich zeigt sich ein weiterer Ausschlag des ersten Datenpunktes. Dieser repräsentiert den Versatz (Offset) um den das Signal, vom Nullniveau ausgesehen, schwingt. Dieser Ausschlag ist bereits in diesem Stadium in voller Höhe ausgeprägt und lässt sich im Vergleich mit den Rohdaten verifizieren. Um den Amplitudenplot mit den tatsächlichen Ausschlägen zu erhalten, wird der Versatz nicht berücksichtigt. Die restlichen Datenpunkte werden um den Mittelpunkt des Datensatzes gespiegelt, womit man die halbe Punkteanzahl mit doppelter Amplitudenhöhe erhält, welche nun den Werten des Eingangssignales entsprechen. Um die Frequenzen zu den Amplitudenausschlägen zu erhalten, wird die Position dieser Ausschläge im Datensatz ermittelt. Der Frequenzvektor selbst ergibt sich aus der Abtastrate als höchste Frequenz mit einer Diskretisierung, die mit der Punkteanzahl des Datensatzes übereinstimmt. Auch dieser Vektor wird gespiegelt, womit die höchste Frequenz der halben Abtastfrequenz entspricht. Durch Einsetzen der Amplitudenposition in den Frequenzvektor erhält man die zugehörige Frequenz, wie der letzte Bildabschnitt verdeutlicht. Analog dazu kann der Phasenwinkel bestimmt werden, welcher sich aus den Imaginär- und Realteilen mit Formel (4.3) bestimmen lässt.

4.3.6 Fensterfunktionen

Fensterfunktionen sind Filter für die Rohdaten, in denen eine einhüllende Funktion mit steigender und fallender Flanke beziehungsweise unterschiedlichen Formen über die zu verarbeitenden Daten gestülpt wird, wie in Abbildung 4.9 ersichtlich.

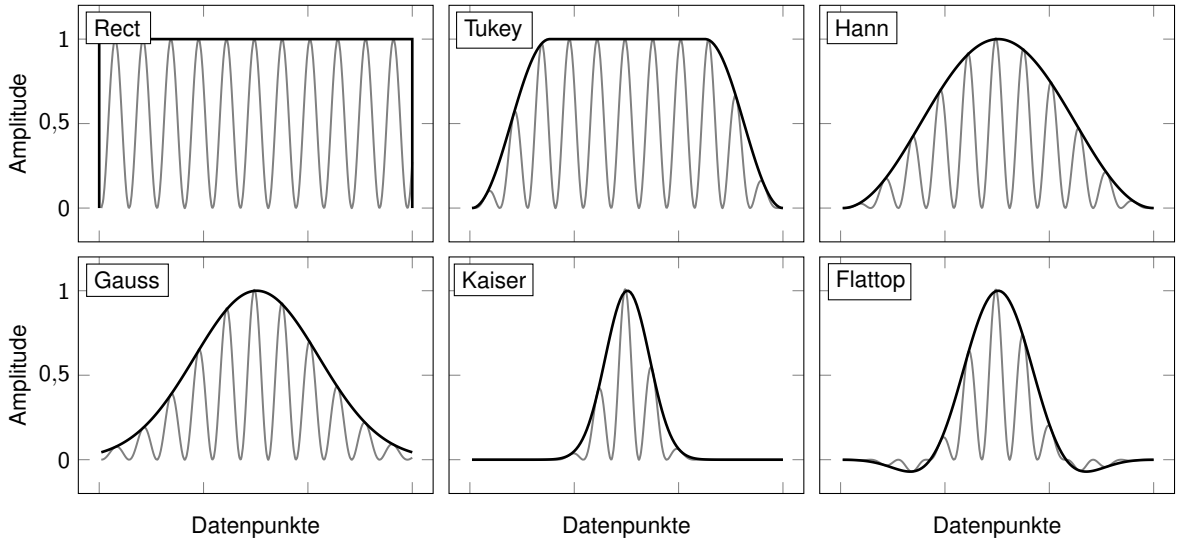

Abbildung 4.9: Verschiedenen Fensterfunktionen zum Filtern der Rohdaten vor der FFT-Analyse. In grau die manipulierten Daten und in schwarz die einhüllende Fensterfunktion.

Dabei gehen die Funktionen von keiner Veränderung der Rohdaten, wie bei der Rect-Funktion, bis hin zur Beschneidung der Daten von wenigen Schwingungen in der Signalmitte, siehe Kaiser-Fenster. Die Mehrheit der Funktionen stellt sich als Glockenkurve dar, wobei sich diese individuell ausprägen. [50]

[50] Vgl. Michael Möser 2018, S. 10.

Diese Ausprägung beeinflusst, welcher Teil der Rohdaten wie stark in die FFT-Analyse einfließt. Dies hat auch den Vorteil, dass zum Beispiel die Einschwingphase eines Systems, bei der noch keine zeitlich konstante Schwingung vorliegt, keinen Einfluss auf die Auswertung hat. Außerdem kann Aliasing-Effekten vorgebeugt werden, wie im nächsten Absatz beschrieben. Der Einfluss dieser Funktionen ist in Abbildung 4.10 illustriert, wo die Signalamplitude eines Messsignales mit unterschiedlichen Fenstern analysiert wurde.

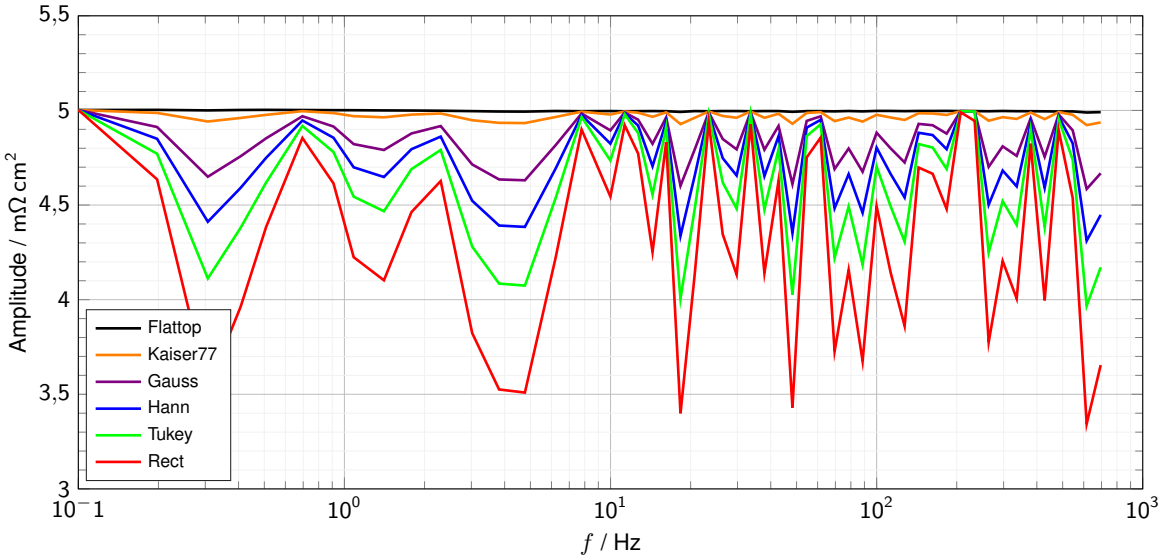

Abbildung 4.10: Vergleich von Fenster-Funktionen anhand des Amplitudenausschlages über einen Frequenzausschnitt mit Messdaten einer Kalibriermessung.

Wenn man bedenkt, dass die Amplitudenhöhe der Messung über diesen Frequenzbereich annähernd konstant ist, kann die beste Fensterfunktion mit dem Flattop schnell ausgemacht werden. Auch das Kaiser-Window liefert ein gutes Ergebnis, unter der Einschränkung, dass der richtige Kaiser-Parameter gewählt wurde. Im Gegensatz zu allen anderen Funktionen, kann durch einen zusätzlichen Parameter, in Form eines Zahlenwertes, die Erscheinung des Fensters beeinflusst werden, wie Abbildung 4.11 verdeutlicht.

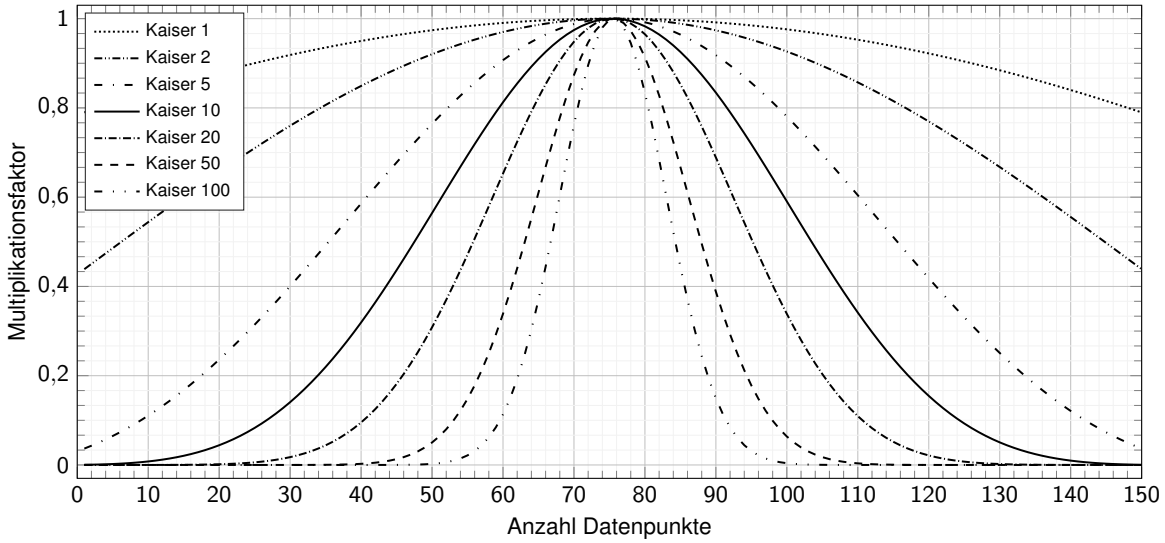

Abbildung 4.11: Vergleich des Kaiser-Windows mit unterschiedlichen Kaiser-Parametern. Je höher der Zahlenwert anwächst, desto enger wird der Bereich im Datensatz der betrachtet wird.

Alle anderen Fensterfunktionen zeigen Abweichungen nach dem gleichen Muster, jedoch mit unterschiedlichen Amplitudenhöhen auf. Wenn man den Quadratwurzelfehler zum Eingangssignal betrachtet, ergibt sich die Rangliste in Tabelle 4.1.

Fensterfunktion	Summe der Fehlerquadrate	Fehler in %
Flattop	0,20	0,08
Kaiser77	1,60	0,61
Gaus	8,48	3,26
Hann	14,15	5,44
Tukey	21,31	8,16
Rect	35,56	13,67

Tabelle 4.1: Empirischer Vergleich der unterschiedlichen Window-Funktionen mittels Quadratwurzelfehler und dem Fehler in Prozent.

Es gibt keine generellen Regeln, wann welches Fenster einzusetzen ist, jedoch gibt es in der Literatur[51] Empfehlungen, wie in Tabelle 4.2 angeführt. Auch hier geht hervor, dass das Flattopfenster für die Analyse der Amplitudenhöhe die besten Ergebnisse liefert, wodurch die empirische Erkenntnis verifiziert wird.

Signalinhalt	Fenster
Sinuswellen oder Kombination aus Sinuswellen	Hann
Sinuswelle (Amplitudengenauigkeit ist wichtig)	Flattop
Schmalband Zufallssignal	Hann
Breitband Zufallssignal (mit Rauschen)	Rect
Eng zusammenliegende Sinuswellen	Rect, Hamming
Unbekannter Inhalt	Hann

Tabelle 4.2: Empfehlung für den Einsatz der Fensterfunktionen.

4.3.7 Spektrale Leckage

Spektrale Leckage tritt dann auf, wenn das Signal keine ganzzahlige Anzahl an Perioden enthält oder die Signalfrequenz nicht im Frequenzvektor der FFT vorhanden ist. Eine ganzzahlige Periodenanzahl kann auf zwei Arten erreicht werden. Indem die Messdauer ein Vielfaches der Periodendauer des Signals beträgt, sofern diese bekannt ist, oder durch das Beschneiden der Rohdaten in der Auswertung. Auch hier muss die Periodendauer beziehungsweise die Punkteanzahl pro Periode bekannt sein, ansonsten ist eine initiale FFT durchzuführen anhand derer man die Periodendauer des Signals ermittelt. Die Leckage rührt daher, dass eine unvollständige Periode durch deren zusätzlichen Energieeintrag die Auswertung und im konkreten die Amplitudenhöhe sowie deren Position im Amplitudenvektor verfälscht, wie Abbildung 4.12 deutlich macht. Ein grafischer Ansatz besagt, dass bei der Aneinanderreihung von unvollständigen Perioden an deren Übergang eine Sprungstelle entsteht, womit sich das Signal nicht mehr stetig fortsetzt und die Analyse verfälscht.[52]

Für Abbildung 4.12 wurde ein synthetisches Signal mit einer Amplitudenhöhe von 2 erzeugt. Bei einer Abtastfrequenz von 10 kHz und einer Signalfrequenz von 200 Hz, enthält der Datensatz 750 Datenpunkte für

[51] Vgl. Michael Cerna, Audrey F. Harvey; 2000, S. 15.
[52] Vgl. Martin Meyer 2017, S. 188.

das obere Signal und 775 unten. Um die Unterschiede in der Amplitudenausprägung besser darstellen zu können, wurde der Amplitudenvektor auf einen Bereich von 30 Punkten eingeschränkt. Bei dem Signal mit voller Periodenanzahl wird die Amplitude in der richtigen Höhe sowie an der korrekten Position ausgegeben. Nur die Punkte neben dieser Position weisen minimale Abweichungen von der Nulllinie auf. Im Gegensatz dazu zeigt die Untersuchung des Signales ohne ganzzahlige Periodenanzahl nur ein unbefriedigendes Ergebnis. Die Amplitude erreicht nur circa zweidrittel ihres Sollwertes und auch die Position ist nicht eindeutig bestimmbar. Weil der höchste Wert für die Positionsfindung herangezogen wird, ergibt sich hier ein Positionsfehler, der sich in der Ausgabe eines falschen Frequenzwertes widerspiegelt. Der breite Auslauf des Amplitudengipfels ist ebenfalls auf die unvollständige Periode zurückzuführen. Da im diskreten Amplitudenvektor der analytisch ermittelte Wert nicht genau getroffen wird, erfährt das Spektrum eine Verschiebung, wodurch sich eine seitliche Streuung um diesen Wertebereich ausbildet [53].

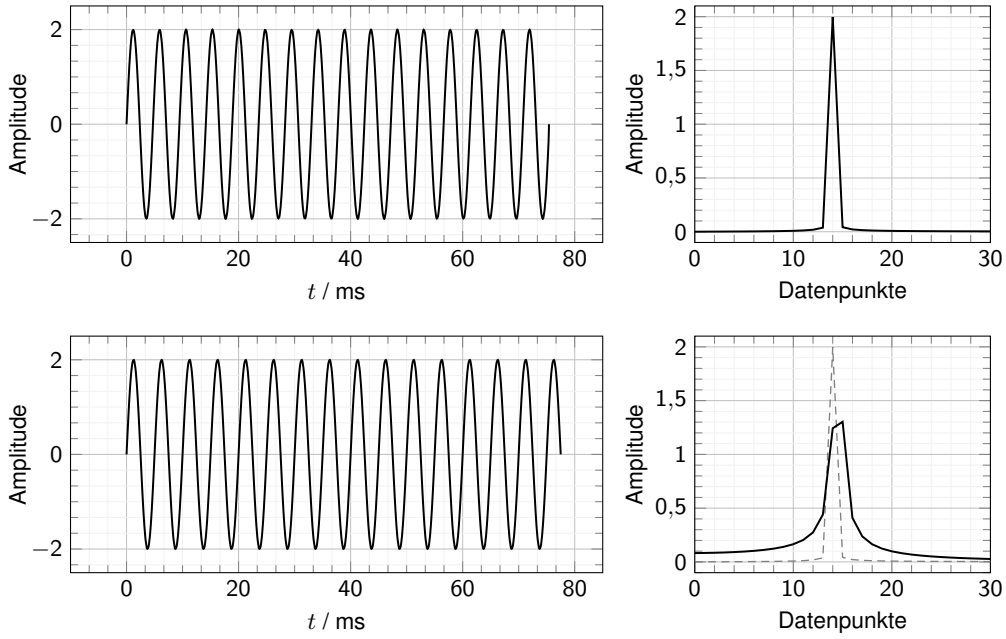

Abbildung 4.12: Einfluss einer nicht ganzzahligen Periodenanzahl auf die Amplitudenhöhe der FFT-Analyse. In der unteren Hälfte ist das Signal um eine halbe Periodendauer länger, beim Amplitudenspektrum handelt es sich zur besseren Darstellung nur um einen Ausschnitt.

Eine weitere Möglichkeit das Signal zu beschneiden ist die Verwendung eines Fensters. Diese reduziert in den gezeigten Beispielen die Bedeutung des Signalanfangs sowie dessen Ende, womit deren Einfluss auf die Analyse reduziert wird. Auch in der periodischen Fortsetzung des Signales zeigt dieses somit einen stetigen Verlauf [54]. Exemplarisch wurde in Abbildung 4.13 das aperiodische Signal mit zwei Fensterfunktionen gewichtet. Wie Abschnitt 4.3.6 gezeigt hat, eignet sich das Flattopfenster besonders für die genaue Amplitudenbestimmung, weshalb dieses gewählt wurde. Zum Vergleich wurde auch ein Fenster, welches im Test schlecht abgeschnitten hat, herangezogen.

Es zeigt sich, dass durch den Einsatz des Tukeyfensters die Amplitude, im Vergleich zum Beispiel ohne Fensterfunktion, an Höhe gewonnen hat und sich somit dem wahren Wert annähert. Auch die Streuung an der Basis erfährt eine merkliche Verbesserung und nähert sich schneller der Nulllinie an. Somit kann man schlussfolgern, dass der Einsatz von Fensterfunktionen die Auswirkungen aperiodischer Signale vermindern kann. Trotzdem ist das Ergebnis nicht zufriedenstellend, zumal die Position wieder nicht exakt getroffen wird und der Fehler in der Amplitudenhöhe immer noch beträchtlich ist. Einen weiteren Beleg für

[53] Vgl. Michael Cerna, Audrey F. Harvey; 2000, S. 13.
[54] Vgl. Martin Meyer 2017, S. 188.

die guten Eigenschaften bei der Amplitudengenauigkeit liefert das Flattop. Dieses gibt die Amplitudenhöhe korrekt wieder, jedoch mit der Einschränkung, dass dieser Wert an gleich zwei Punkten vorliegt. Somit kann die Position, rein optisch, nicht eindeutig bestimmt werden. Im Datensatz ist jedoch ein Unterschied an der dritten Kommastelle feststellbar, womit auch hier die falsche Position ausgegeben wird. Im Gegensatz zum vorherigen Fenster bildet sich eine insgesamt breitere Form aus, die jedoch ähnlich schnell gegen die Nulllinie konvergiert. Diese Beispiele zeigen, dass die Genauigkeit der FFT-Analyse maßgeblich davon abhängt, ob der Datensatz auf eine volle Periodenanzahl beschränkt wird.

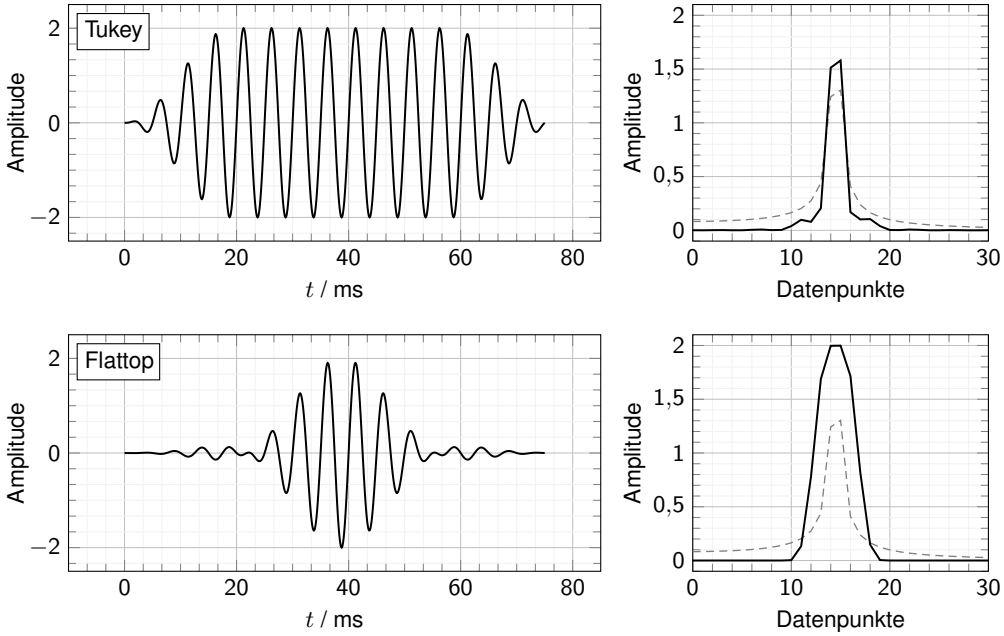

Abbildung 4.13: Einfluss von Fensterfunktionen auf aperiodische Signale.

Der zweite Faktor für die Genauigkeit der FFT-Analyse ist die Schrittweite des Frequenzvektors. Wie in Abschnitt 4.3.5 beschrieben, hängt diese von der Abtastrate und der Anzahl der Datenpunkte ab. Je länger die Messdauer eines Signales im Verhältnis zur Abtastrate ist, desto feiner ist die Schrittweite im Frequenzvektor und desto kleiner wird der absolute Fehler, wie Abbildung 4.14 zeigt.

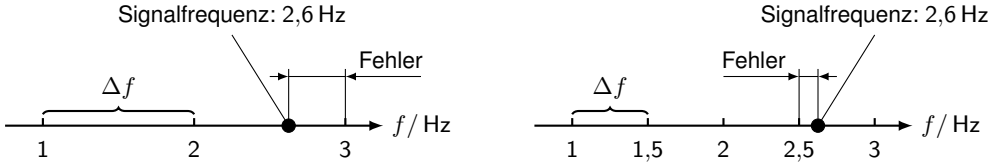

Abbildung 4.14: Auswirkungen der Schrittweite im Frequenzvektor bei Signalfrequenzen zwischen den absoluten Frequenzwerten.

Nur das Erhöhen der Abtastrate bei gleicher Messdauer ändert jedoch nichts an der Schrittweite. Das liegt daran, dass die Anzahl der Abtastpunkte im Zeitbereich proportional zur Abtastfrequenz ist. Das Signal wird zwar feiner aufgelöst, jedoch muss im Frequenzvektor ein höherer Endwert erreicht werden, weshalb die Schrittweite konstant bleibt. Dies zeigt auch der empirische Versuch im oberen Diagramm der Abbildung 4.15, wo die Abtastrate von 50 Hz sukzessive auf 250 kHz erhöht wurde und sich die Schrittweite Δf nicht verändert hat. In der zum Diagramm nebenstehenden Grafik ist dies nochmals illustriert. Es muss im Frequenzvektor ein doppelt so hoher Wert erreicht werden bei Verdoppelung der Punkteanzahl, womit die Schrittweite Δf ident bleibt. Um diese zu minimieren und damit die Auflösung des Frequenzvektors zu erhöhen, muss die Messdauer und damit die Anzahl der Datenpunkte gesteigert werden. Dies ist im zweiten

Diagramm 4.15 ersichtlich, wo sich die Schrittweite Δf mit zunehmender Periodenanzahl im Messsignal asymptotisch dem Nullwert und damit einer theoretisch unendlichen Auflösung annähert.

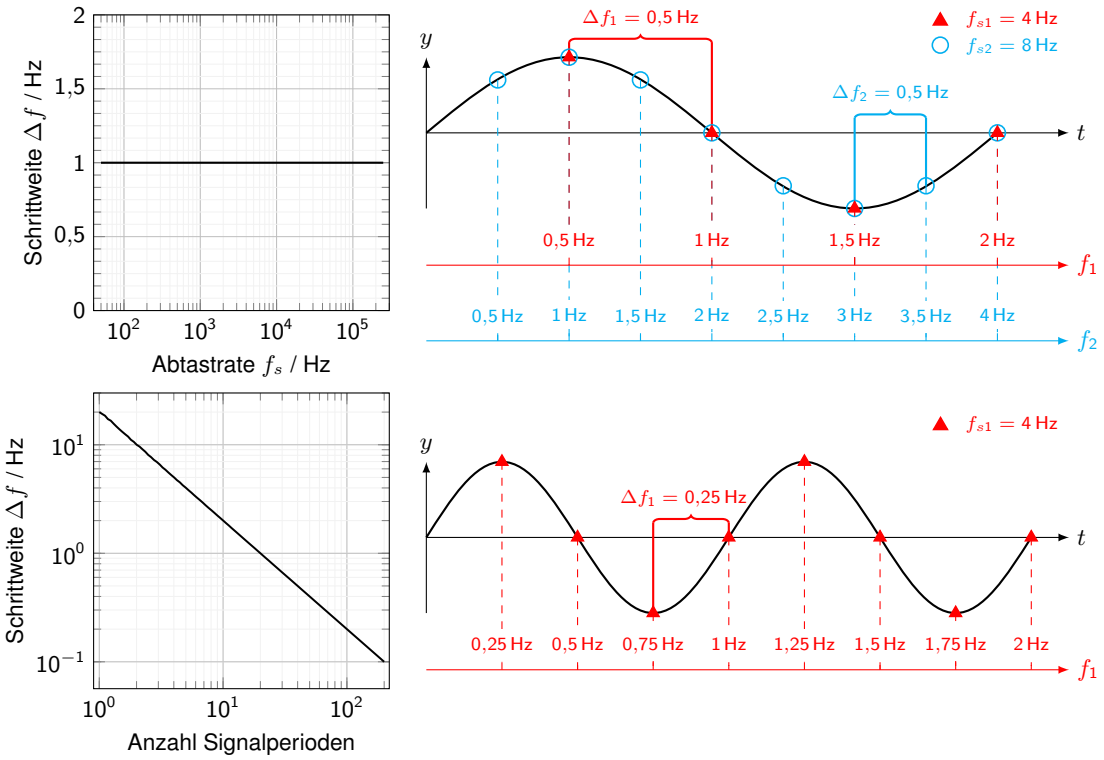

Abbildung 4.15: Verhältnis von Abtastrate und Periodenanzahl zu Schrittweite.

Neben der unvollständigen Periode ist der Fehler im Frequenzvektor hauptverantwortlich für die spektrale Leckage. Die Auswirkungen auf den Amplitudenvektor sind in Abbildung 4.16 ersichtlich. Hierbei wurde ein Eingangssignal mit einer Amplitude von 2 und einer Frequenzschrittweite von $\Delta f = 2\,\text{Hz}$, bei einer Abtastrate von 100 Hz, verwendet. Die Eingangssignalfrequenz wird aus dem Frequenzvektor mit 20 Hz (grau) entnommen, wodurch der Wert in der FFT-Analyse genau getroffen wird und keine Leckage auftritt. Um den größtmöglichen Fehler dieses Phänomens zu illustrieren, wird für die Signalfrequenz zusätzlich 19 Hz (schwarz) definiert, welche genau zwischen zwei absoluten Frequenzwerten im Frequenzvektor liegt. Zur besseren Darstellung wird wieder nur der Bereich im Amplitudenvektor rund um den Amplitudenausschlag gezeigt.

Es zeigt sich auch hier wieder, dass das Flattop die beste Option hinsichtlich Amplitudengenauigkeit ist. Im Vergleich der beiden Linien zeigt sich der Frequenzfehler anhand der Flachstelle an der Spitze. Aber auch wenn die Frequenz genau getroffen wird, bildet sich ein Plateau aus, jedoch mit einer Spitze, die den Frequenzwert des Signales widerspiegelt. Ähnlich gut bei der Ermittlung des Amplitudenwertes ist das Kaiserfenster. Dieses besitzt zwar die größte Streuung an der Basis, weist aber beim Amplitudenwert nur einen geringen Fehler auf. Analog zum Flattop bildet sich auch hier wieder ein Plateau mit einer Spitze aus, wenn die Frequenz genau getroffen wird. Das Gaussfenster und Hannfenster zeigen ähnliche Ergebnisse beim Wert der Amplitude und unterscheiden sich hauptsächlich in ihrer Ausprägungsform. Im Vergleich zu den beiden besten Fenster, bildet sich jedoch eine deutlichere Spitze aus, wenn die Frequenz getroffen wird. Der Datensatz ohne Fensterfunktion zeigt auch hier wieder die schlechtesten Ergebnisse mit dem größten Amplitudenfehler. Stimmt die Signalfrequenz jedoch mit einer im Frequenzvektor überein, so bildet sich genau an dieser Position ein einzelner Peak aus. Wenn kein Fehler zwischen Signalfrequenz und absolutem Frequenzwert vorliegt, erkennen alle Fensterfunktionen zuverlässig die Amplitudenhöhe. Aus Abbildung

4.16 kann weiter abgleitet werden, dass die Bildung eines Plateaus an der Spitze ein Indikator für spektrale Leckage zu sein scheint, da dieses Phänomen bei allen Untersuchungen auftritt. Auch mit einer höheren Punktedichte und damit geringeren Schrittweite ändert sich daran nichts, solang die Signalfrequenz immer genau zwischen zwei absoluten Frequenzen im Frequenzvektor liegt.

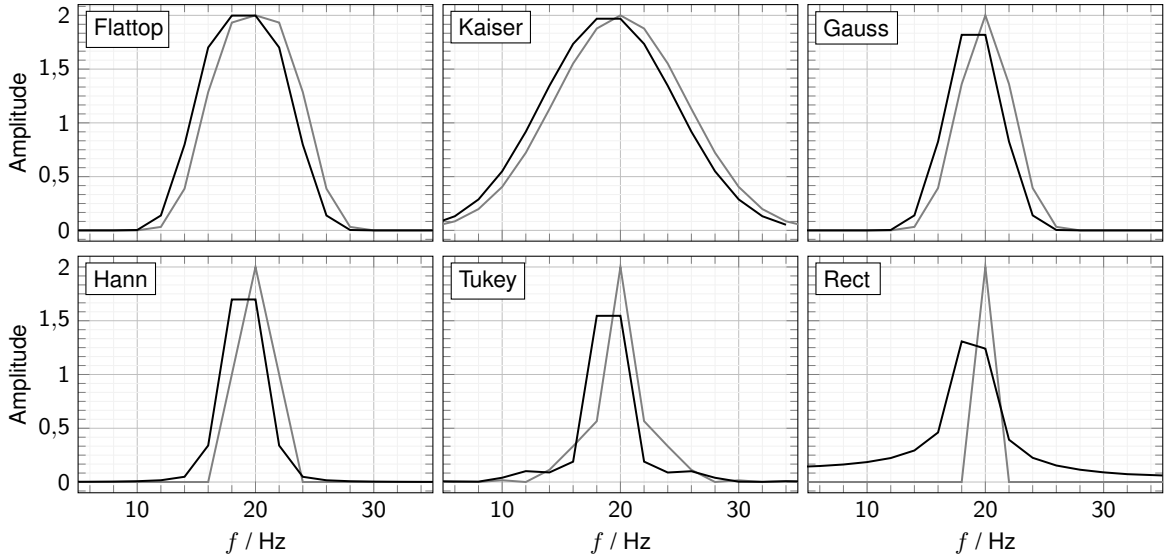

Abbildung 4.16: Vergleich der Amplitudenausprägung mit verschiedenen Fensterfunktionen. Die graue Linie zeigt die Ausprägung, wenn die Signalfrequenz einem Frequenzwert im Frequenzvektor entspricht. Die schwarze Linie illustriert das Ergebnis, wenn die Signalfrequenz (schwarz 20 Hz und grau 19 Hz) genau zwischen zwei Frequenzwerten des Frequenzvektors liegt.

In Abbildung 4.17 ist der Verlauf der Signalstärke über die zunehmende Abtastrate aufgetragen. Wie oberhalb näher ausgeführt, bleibt die Schrittweite bei gleicher Messdauer und steigender Abtastrate gleich.

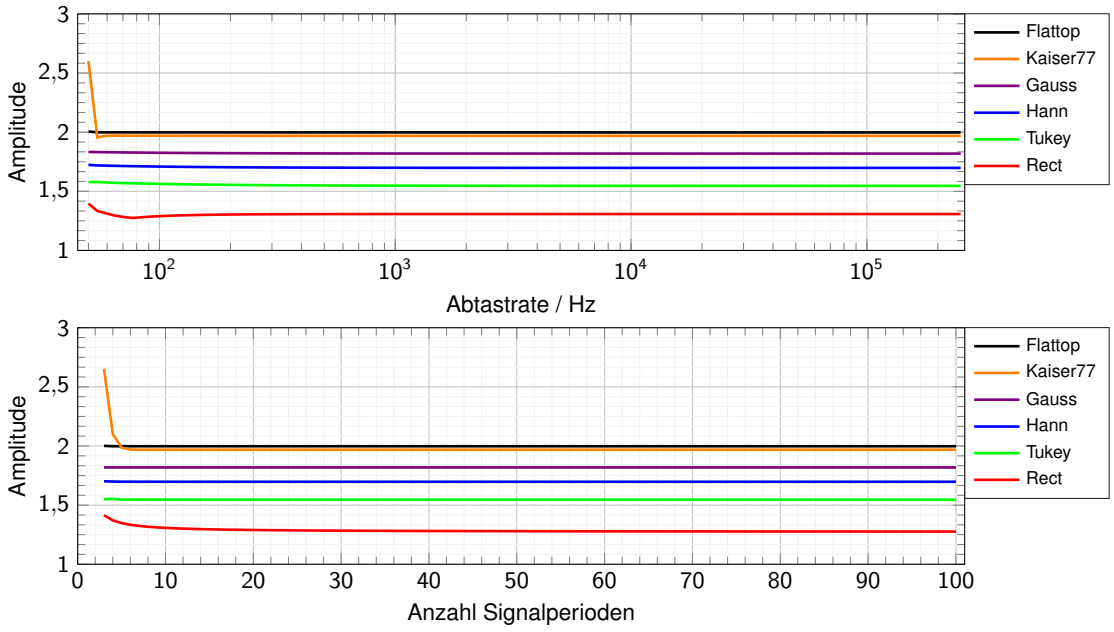

Abbildung 4.17: Verlauf der Signalstärke über ein Frequenzband der Abtastrate mit verschiedenen Fensterfunktionen und einer Signallänge von 10 Perioden. Analog dazu dieselbe Untersuchung mit steigender Anzahl an Signalperioden und einer konstanten Abtastrate von 250 Hz. Das Eingangssignal hat jeweils eine Amplitude von 2 und die Signalfrequenz liegt immer genau zwischen zwei absoluten Frequenzwerten im Frequenzvektor.

Deshalb stagnieren auch die Amplitudenwerte, bis auf den sehr niedrigen Frequenzbereich nahe der Nyquist-Shannon Grenze. Auch die Steigerung der Messdauer und damit die Anzahl der Datenpunkt hat darauf keine Auswirkung, weil die Signalfrequenz immer noch bei jeder einzelnen Periodenanzahl genau zwischen zwei Frequenzwerten liegt.

Wenn die Signalfrequenz nicht mehr für jeden Datenpunkt genau zwischen zwei Frequenzen gesetzt wird, ergibt sich ein anderes Bild wie Abbildung 4.18 zeigt. Die Amplitude beträgt wieder 2 bei einer Signalfrequenz von 19 Hz und einer Abtastrate von 250 Hz. Diese Signalfrequenz liegt bei einer Periodenanzahl von 10 genau zwischen zwei Frequenzwerten, was sich auch im Plot widerspiegelt. Da nun die Signalfrequenz fixiert ist und die Auflösung des Frequenzvektors durch die steigende Anzahl der Messpunkte immer feiner wird, ergibt sich das periodische Verhalten. So nähert sich der Frequenzwert des Signales immer weiter einem absoluten Frequenzwert an, da die Schrittweite im Vektor abnimmt, bis diese ident sind. Da die Schrittweite aber immer kleiner wird, entfernt sich die Signalfrequenz wieder von den Absolutwerten, bis diese wieder genau zwischen zwei Frequenzwerten liegt. Dies wiederholt sich periodisch, in diesem Fall bei jeder Zunahme um 20 Perioden, wodurch sich die Wellenform im Diagramm ergibt.

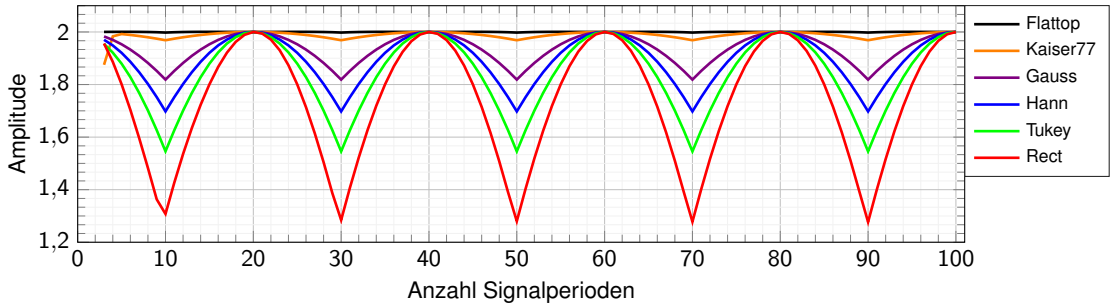

Abbildung 4.18: Periodisches Verhalten der Signalstärke bei zunehmender Anzahl der Signalperioden und fixer Signalfrequenz für verschiedene Fensterfunktionen.

Dieses Phänomen lässt sich am besten an der Ausprägung der Amplitude nachvollziehen. Im Gegensatz zu Abbildung 4.18 ändert sich in Abbildung 4.19 nicht der Frequenzvektor, sondern die Signalfrequenz wandert durch einen kleinen Bereich, die Mechanismen bleiben jedoch die gleichen.

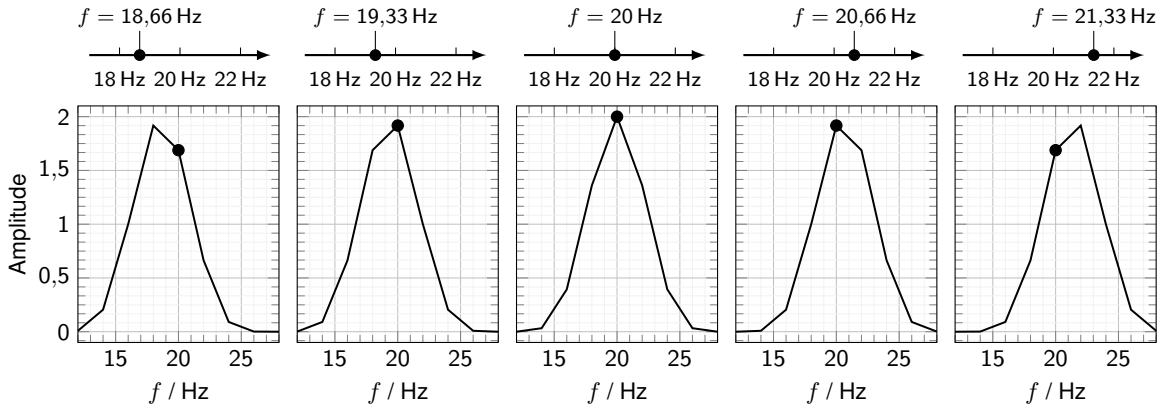

Abbildung 4.19: Ausprägung der Amplitude zwischen den absoluten Frequenzschritten mit dem Gaussfenster bei unterschiedlichen Signalfrequenzen.

Da der erste Frequenzpunkt näher an den 18 Hz liegt als an den 20 Hz, erreicht auch im Diagramm die Amplitude ihren Maximalwert bei 18 Hz. Im nächsten Abschnitt liegt der Frequenzwert näher an den 20 Hz,

weshalb sich nun die Amplitude im Diagramm dorthin verschiebt. Liegt die Signalfrequenz genau auf dem Wert im Frequenzvektor, zeigt sich ein eindeutiges Ergebnis mit dem korrekten Amplitudenwert. Entfernt sich die Frequenz wieder von diesem Wert, nimmt auch die Amplitudenhöhe wieder ab. Verfolgt man den Messpunkt bei 20 Hz in den Diagrammen, zeigt sich das periodische Verhalten wie in Abbildung 4.18, dass durch die finite Anzahl an Frequenz im Frequenzvektor hervorgerufen wird. Alle Untersuchungen sowie die Empfehlung aus der Literatur liefern das gleiche Ergebnis. Für die Bestimmung der Amplitude und deren Position im Vektor ist das Falltop am besten geeignet.

4.4 Fitten von Daten

Aufgrund der Frequenzabhängigkeit der Messkette sowie der daraus resultierenden Nichtlinearität, müssen die Messdaten um den eingebrachten Fehler korrigiert werden. Hierfür werden Kalibrierungsmessungen durchgeführt, wie in Kapitel 7 näher beschrieben. Da diese Kalibriermessungen nicht zwangsläufig mit den Frequenzen der späteren Versuchsmessungen am Stack übereinstimmen und eine sehr hohe Dichte der Messpunkte den Mess- und Auswerteaufwand sehr stark steigern, gibt es mehrere Optionen die Lücken zwischen den Punkten mathematisch zu erschließen. Diese haben alle gemein, dass versucht wird aus dem Datensatz eine lineare oder nichtlineare Modellfunktion zu generieren. Anhand dieser kann dann für jeden beliebigen Eingabewert ein Funktionswert ausgegeben werden, ohne einen hochauflösenden Datensatz zu benötigen. Anstatt des gesamten Datensatzes müssen nur noch die Funktionsparameter gespeichert werden. Dabei wird die Problemstellung in drei Szenarien unterteilt. [55]

1. Die strukturelle Form der Modellfunktion ist bekannt und es müssen nur noch die jeweiligen Parameter ermittelt werden.
2. Die strukturelle Form ist unbekannt, kann aber aufgrund des Datenverlaufes oder durch Kenntnis des gemessenen Systems abgeschätzt werden.
3. Die strukturelle Form ist unbekannt und auch sonst lassen sich keine Annahmen treffen.

Im Umfang dieser Arbeit werden nur die ersten beiden Szenarien behandelt, da diese für die vorliegende Problemstellung ausreichen.

4.4.1 Lineare Interpolation

Die lineare Interpolation geht immer davon aus, dass sich die zeitliche Entwicklung des gemessenen Systems innerhalb zweier Messpunkte linear verhält. Somit kann jeder beliebiger Punkt zwischen zwei Stützpunkten einfach über eine Umformung der Geradengleichung

$$y_n = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \, (x_n - x_1) \tag{4.20}$$

berechnet werden. Der dabei entstehende Fehler ist stark von der Punktedichte und damit vom Verhältnis von Abtastrate zur zeitlichen Änderung des Signales abhängig. Deshalb muss die Abtastrate zur Signaldynamik angepasst werden, wie Abbildung 4.20 deutlich macht.

Bei einer ungünstigen Kombination aus geringer Punktedichte und stark schwankendem Signal, kann man durch die Interpolation eine Fehlergröße in die Auswertung einbringen. Grund dafür ist unter anderem

[55] Vgl. Achim Zielesny 2016, S. 62–63.

auch, dass der Signalverlauf vor und nach den beiden Datenpunkten nicht miteinbezogen wird und somit die Signaltendenz nicht berücksichtigt wird.

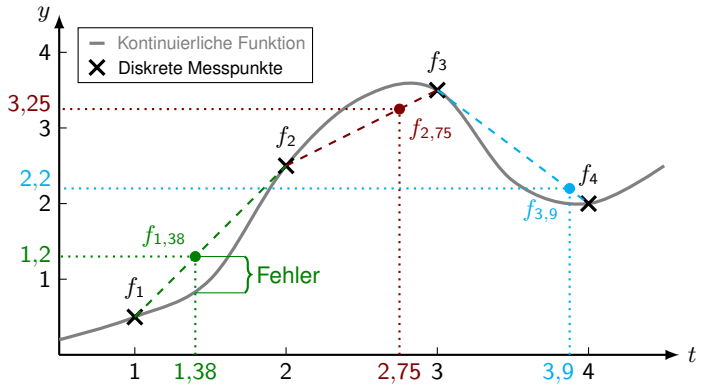

Abbildung 4.20: Lineare Interpolation zwischen Punkten und der daraus resultierende Fehler.

4.4.2 Polynom Regression

Ist die Modellfunktion bekannt, liegt die Aufgabe in der Findung der Funktionsparameter. Dies erfolgt meistens nach einem Regressionsverfahren, dessen Ziel ist es, die Abweichung zwischen Funktion und Daten zu minimieren. Zwei der bekanntesten Fehlermaße sind der mittlere absolute Fehler E_1 und der Quadratwurzelfehler E_2

$$E_1(f) = \frac{1}{n}\sum_{k=1}^{n}|f(x_k) - y_k|, \qquad E_2(f) = \left[\frac{1}{n}\sum_{k=1}^{n}|f(x_k) - y_k|^2\right]^{1/2}, \tag{4.21}$$

wobei y_k für die Datenpunkte steht und n die Anzahl derer beschreibt. Der Vorteil dieser Verfahren ist der geringe Rechenaufwand. Angenommen, man möchte den Trend eines Datensatzes mit einer Linie bestimmen, so kann man diese mit der Geradengleichung wie folgt beschreiben

$$f(x) = m\,x + c. \tag{4.22}$$

Um ein Minimum der Fehlerfunktion zu erhalten, muss nur die Summe minimiert werden, womit gilt

$$E_2(f) = \sum_{k=1}^{n}|f(x_k) - y_k|^2 = \sum_{k=1}^{n}(m\,x_k + c - y_k)^2\,. \tag{4.23}$$

Da die Funktion zwei Unbekannte enthält, muss diese nach den beiden Variablen m und c partiell abgeleitet werden

$$\frac{\partial E_2}{\partial m} = 0: \quad \sum_{k=1}^{n} 2\,(m\,x_k + c - y_k)\,x_k = 0 \quad \Longrightarrow \quad m\sum_{k=1}^{n}x_k^2 + c\sum_{k=1}^{n}x_k - \sum_{k=1}^{n}x_k\,y_k = 0, \tag{4.24}$$

$$\frac{\partial E_2}{\partial c} = 0: \quad \sum_{k=1}^{n} 2\,(m x_k + c - y_k) = 0 \quad \Longrightarrow \quad m\sum_{k=1}^{n}x_k + c\,n - \sum_{k=1}^{n}y_k = 0. \tag{4.25}$$

Das lineare Gleichungssystem kann auch als 2×2 Matrix angeschrieben werden

$$\begin{bmatrix} \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k \\ \sum_{k=1}^{n} x_k & n \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} x_k\, y_k \\ \sum_{k=1}^{n} x_k \end{bmatrix}. \tag{4.26}$$

Durch das Lösen des Gleichungssystems erhält man schlussendlich die beiden Koeffizienten, welche die Gerade mit dem geringsten Quadratwurzelfehler beschreiben. [56]

Diese Vorgangsweise kann analog auch bei Polynomen höheren Grades angewendet werden. Dadurch ist es möglich Daten nicht nur zu interpolieren, sondern auch den Bereich außerhalb des Datensets zu extrapolieren. Hierbei ist jedoch entscheidend, dass der Grad des Polynoms nicht zu niedrig gewählt wird, wie Abbildung 4.3 verdeutlicht.

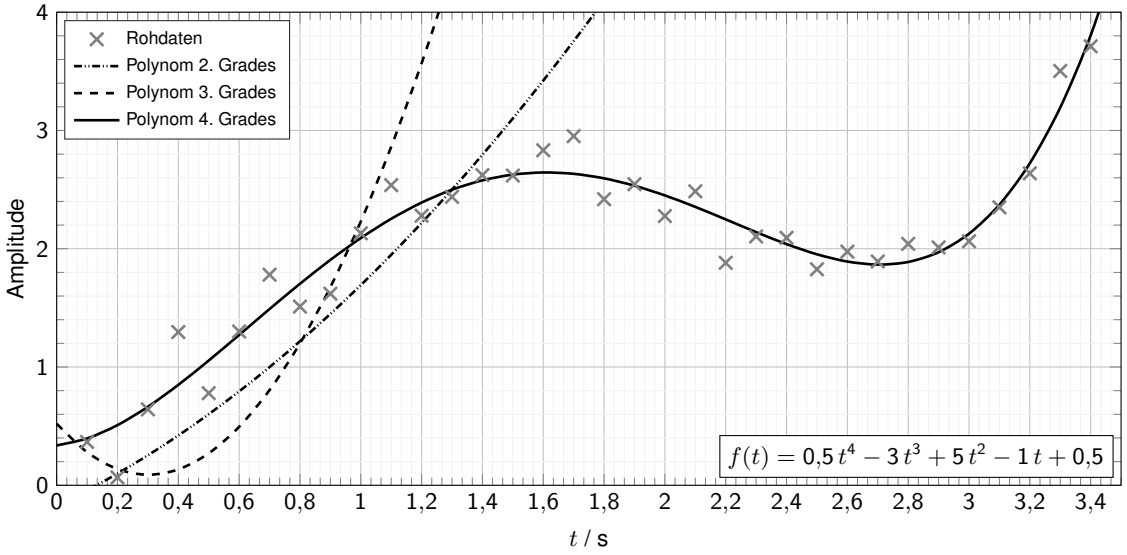

Abbildung 4.21: Fitten von Daten mit Polynom-Funktionen verschiedenen Grades.

Die Rohdaten, die der Abbildung 4.21 zu Grunde liegen, folgen einem Polynom vierten Grades mit addierten Zufallszahlen. Die Polynome zweiten und dritten Grades können der Funktion höheren Grades naturgemäß nicht folgen und sind deshalb ungeeignet diese zu beschreiben. Erst wenn derselbe Grad erreicht wird, können die Daten mit dem Polynom beschrieben werden. Deshalb müssen für dieses Verfahren Informationen über das System vorliegen, oder der Grad wird so lange erhöht, bis das Polynom den Daten folgen kann, sofern diese mit einem Polynom beschrieben werden können.

4.4.3 Spline Regression

In der Polynom Regression wird eine Funktion mit bestimmtem Grad einem Datensatz, mittels Minimierung des Fehlers zwischen Funktion und Daten, angenähert. Dadurch muss die Funktion nicht zwangsläufig einen Datenpunkt schneiden, wie auch Abbildung 4.21 zeigt. Soll das Polynom aber durch die Datenpunkte verlaufen, muss dessen Grad gleich der Anzahl n der Datenpunkte sein

$$p_n(x) = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + \cdots + a_n\,x^n \tag{4.27}$$

[56] Vgl. Steven L. Brunton, J. Nathan Kutz 2017, S. 136–140.

Dadurch hat das Polynom $n+1$ Unbekannte und braucht somit auch ebenso viele Gleichungen, damit ein lineares Gleichungssystem aufgestellt werden kann

$$\begin{array}{ccc} (x_0,\, y_0) & \longrightarrow & y_0 = a_0 + a_1\, x_0 + a_2\, x_0^2 + a_3\, x_0^3 + \cdots + a_n\, x_0^n \\ \vdots & & \vdots \\ (x_n,\, y_n) & \longrightarrow & y_n = a_0 + a_1\, x_n + a_2\, x_n^2 + a_3\, x_n^3 + \cdots + a_n\, x_n^n \end{array} . \tag{4.28}$$

Daraus resultiert aber auch, dass die Funktion $n-2$ Wendepunkte besitzt, was speziell an den Rändern der Datenreihe zu so genanntem Polynomwackeln (engl.: Polynomial wiggle) führt. Dieses Phänomen ist in Abbildung 4.21 ersichtlich. Auch mit diesen Ausschlägen ist der Fehler an den Punkten gleich Null, da die Funktion trotzdem noch durch alle Punkte verläuft. Problematisch ist jedoch die Interpolation zwischen den Punkten an den Rändern, da die Funktion hier offensichtlich nicht dem Trend des Datensatzes folgt. Die Datenreihe wird zwar abseits der Ränder sehr gut beschrieben, die extremen Ausschläge an den ersten und letzten Punkten machen diese Methode jedoch unbrauchbar. [57]

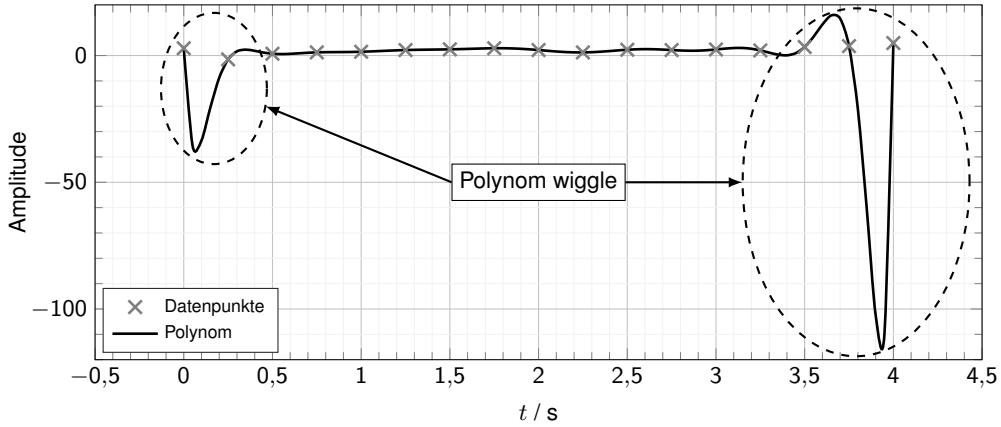

Abbildung 4.22: Polynom dessen Grad der Anzahl der Datenpunkte entspricht und dem resultierendem Polynomwackeln.

Um diesem Phänomen vorzubeugen, folgt die Spline-Interpolation einer anderen Strategie. Hierbei wird immer nur ein kubisches Polynom zwischen drei Punkten gefitted. Ein Polynom dritten Grades hat aber auch vier unbekannte, womit für jedes Polynom vier Randbedingungen aufzustellen sind. Diese Bedingungen lauten wie folgt:

1. Das Polynom muss durch den Start- und Endpunkt des Datenabschnittes verlaufen.
2. Am Übergang von einem Abschnitt der Datenreihe zum nächsten müssen die Punkte übereinander liegen.
3. Die erste Ableitung und damit die Steigung am Punkt des Überganges von einem Abschnitt zum Nächsten müssen gleich sein.
4. Auch die zweite Ableitung und damit die Krümmung muss am Punkt des Überganges ident sein.

Werden diese Bedingungen erfüllt, erhält man eine kontinuierliche und glatte Funktion. Trotzdem tritt ein Problem an den Rändern auf, da hier nicht alle Randbedingungen erfüllt werden können. Es fehlen exakt zwei Bedingungen, die einfach vorgegeben werden, wie etwa die Krümmung an den Endpunkten ist

[57] Vgl. J. Nathan Kutz 2015.

Null. Der Fit erscheint zwar wie eine kontinuierliche Funktion, ist aber in Wahrheit nur eine Aneinanderreihung von individuellen Polynomen. Somit ist eine Interpolation innerhalb der Datenreihe möglich, eine Extrapolation wie bei der Polynom Regression jedoch nicht. [58]

Eine solche Spline-Interpolation ist in Abbildung 4.23 illustriert. Es handelt sich um die gleichen Rohdaten wie in Abbildung 4.21. Erst in der Vergrößerung ist deutlich zu erkennen, dass der Spline von der Verbindungslinie abweicht und im Gegensatz zu dieser einen glatten und kontinuierlichen Verlauf aufweist. Der Spline ist somit ideal für Daten, die keiner bekannten mathematischen Funktion folgen.

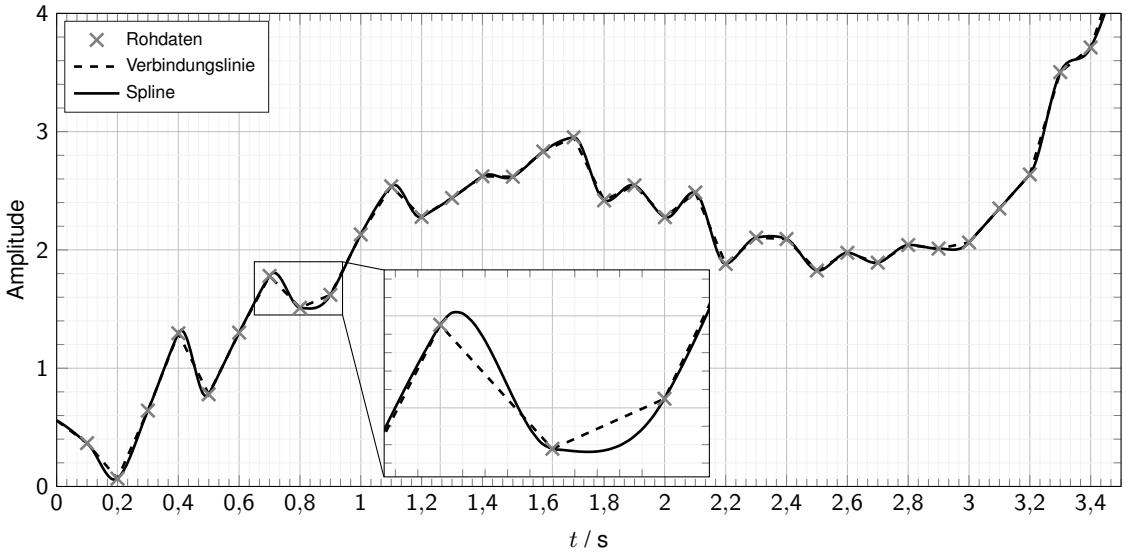

Abbildung 4.23: Fitten von Daten mittels Spline-Funktion. Werden nur die Abtastpunkte eingesetzt erhält man diese exakt wieder, zwischen den Messdaten wird über die Spline-Funktion interpoliert.

[58] Vgl. J. Nathan Kutz 2015.

5 ELEKTROCHEMISCHE IMPEDANZSPEKTROSKOPIE

Die elektrochemische Impedanzspektroskopie (EIS) dient, wie der Name schon impliziert, der Charakterisierung von elektrochemischen Systemen. Klassische Messgrößen in der Elektrochemie sind etwa Strom, elektrische Ladung oder elektrochemische Potenziale als Funktionen der Zeit. Im Gegensatz dazu ist EIS eine Frequenzanalyse bei konstantem Potenzial und produziert auswertbare Information bei jeder gemessenen Frequenz. Als Ergebnis erhält man komplexe Zahlenwerte, wobei jeder Messpunkt drei Informationen enthält, nämlich den Real- und Imaginärteil sowie die Frequenz bei der die Messung durchgeführt wurde. Außerdem bilden sich über ein gemessenes Frequenzband Impedanzkurven aus, deren Form weitere wichtige Informationen über das Prüfobjekt preisgeben. Trotz der Informationsfülle kann EIS nicht alle anderen Charakterisierungsmethoden ersetzen. Es bietet sich jedoch ein sehr breites Feld an Anwendungsmöglichkeiten: [59]

1. Grenzflächenprozesse: Redoxreaktion an Elektroden, Adsorption und Elektrosorption, Kinetik homogener Reaktionen in Lösungen in Verbindung mit Redoxprozessen sowie erzwungener Stoffaustausch.
2. Geometrische Effekte: linearer, sphärischer, zylindrischer Stoffaustausch, volumenbegrenzte Elektroden, Bestimmung des Lösungswiderstandes oder poröse Elektroden.
3. Anwendungen in den Bereichen Energiequellen (Batterien, Brennstoffzellen, Superkondensatoren, Membranen), Korrosion, elektrokatalytische Reaktionen (Wasserelektrolyse, Cl_2-Entwicklung), leitfähige Polymere, Halbleiter und viele mehr.

Das größte Einsatzgebiet für die EIS in der Wasserstofftechnik ist die Materialforschung. Der Großteil der Publikationen in der PEM-Technologie beschäftigt sich hierbei mit der Brennstoffzelle. Bei der PEM-Elektrolyse steht ebenfalls die Materialforschung im Vordergrund, jedoch werden fast ausschließlich Einzelzellen untersucht [60] [61]. Einige wenige beschäftigen sich auch mit der Vermessung von Stacks, wobei nur mehrere ausgewählte Einzelzellen und nicht die gesamten Stacks vermessen werden [62] [63]. Somit stellt die Vermessung eines kompletten Stacks inklusive aller Zellen ein Novum dar.

5.1 Wechselstromtechnik und Impedanzmessung

Bei der Elektrolyse liegt der Fokus eigentlich auf der Gleichstromtechnik, da der Stack mit einer Gleichstromquelle versorgt wird. Gleichstrom zeigt kein periodisches Verhalten und nimmt auch keine negativen Werte an, womit sich keine Phasenverschiebung zwischen Strom und Spannung ergibt. Da in der Wechselstromtechnik jedoch eine Abhängigkeit der Bauteile mit der Signalfrequenz vorliegt, werden die elektrotechnischen Größen in komplexer Form angegeben.

[59] Vgl. Andrzej Lasia 2014, S. 1–3.
[60] Vgl. K. Elsøe, L. Grahl-Madsen, G. G. Scherera, J. Hjelm, M. B. Mogensen; 2017, S. 1419–1426.
[61] Vgl. F.N. Khatib, Tabbi Wilberforce, Oluwatosin Ijaodola, Emmanuel Ogungbemi, Zaki El-Hassan, A. Durrant, J. Thompson, A.G. Olabi; 2019, S. 1–14.
[62] Vgl. S. Siracusano, A. Di Blasi, V. Baglio, G. Brunaccini, N. Briguglio, A. Stassi, R. Ornelas, E. Trifoni, V. Antonucci, A.S. Arico; 2011, S. 3333–3339.
[63] Vgl. Shucheng Sun, Zhigang Shao, Hongmei Yu, Guangfu Li, Baolian Yi; 2014, S. 515–520.

Der Wechselstromwiderstand wird als komplexe Impedanz $\underline{Z}$, in Abhängigkeit der Frequenz, bezeichnet und ergibt sich wie folgt

$$\underline{Z} = \frac{\underline{u}}{\underline{i}} = R + \mathrm{j}X. \tag{5.1}$$

Wobei R als Realteil, an dem keine Phasenverschiebung auftritt, und X als Imaginärteil mit einer Phasenverschiebung von +90° definiert ist. Wie Abbildung 5.1 zeigt, ergibt sich der reelle Scheinwiderstand Z somit aus dem Betrag der komplexen Impedanz $\underline{Z}$.

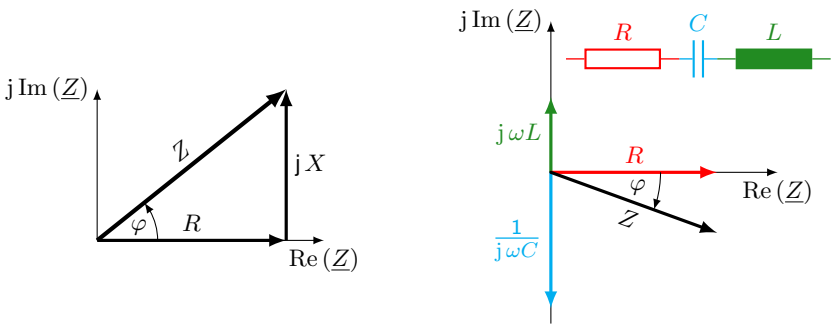

Abbildung 5.1: Links der Scheinwiderstand als Betrag von Real- und Imaginärteil des komplexen Widerstandes. Rechts der Scheinwiderstand eines RLC-Gliedes.

Der resultierende Winkel der Phasenverschiebung φ ist ein Maß für den zeitlichen Versatz, der durch den Imaginäranteil entsteht. Bei einer Schaltung die sowohl induktive- als auch kapazitive Komponenten enthält, werden die imaginären Anteile aufsummiert, wodurch sich der resultierende imaginäre Anteil ergibt. Da der Widerstandswert im Zeitbereich je nach Betrachtungszeitpunkt variiert, wird dieser in den Frequenzbereich überführt

$$\underline{Z}(\mathrm{j}\,\omega) = \frac{\mathcal{F}\{\underline{u}\}}{\mathcal{F}\{\underline{i}\}} = \frac{U(\mathrm{j}\,\omega)}{I(\mathrm{j}\,\omega)}. \tag{5.2}$$

Da die zeitliche Verschiebung und damit das Verhältnis von Real- zu Imaginärteil essentiell in der Impedanzspektroskopie ist, wird kurz das Frequenzverhalten der Grundschaltelemente thematisiert.

Der Ohmsche Widerstand zeigt sowohl in der Gleich- als auch in der Wechselstromtechnik dasselbe Verhalten und beeinflusst die Phasenverschiebung nicht, weshalb gilt

$$Z = R = \frac{u_R}{i_R} \qquad \text{und} \qquad \varphi = 0°. \tag{5.3}$$

Im Gegensatz dazu bringt der Kondensator einen kapazitiven Blindwiderstand in Abhängigkeit der Signalfrequenz ω und der Kapazität C ins Signal ein

$$X_C = -\frac{1}{\omega\, C} \qquad \text{und} \qquad \varphi = \varphi_u - \varphi_i = -90°. \tag{5.4}$$

Der Kondensator wirkt wie ein Widerstand, der ständig Energie aufnimmt, speichert und diese wieder abgibt. Die Energie wird ohne Wirkung hin und her geschoben, weshalb man auch von einem Blindwiderstand spricht. Dadurch wird eine Phasenverschiebung zwischen Strom und Spannung erzeugt die +90° beträgt, womit der Strom der Spannung voreilt. Je höher die Frequenz, desto niederohmiger wird der Blindwiderstand.

Das Verhalten einer Spule ist reziprok zum Kondensator und wird neben der Frequenz ω durch die Induktivität L beschrieben [64]

$$X_L = \omega\, L \qquad \text{und} \qquad \varphi = \varphi_u - \varphi_i = 90°. \tag{5.5}$$

[64] Vgl. Patrick Schnabel 2014, S. 105–110.

Jedoch ist die Phasenverschiebung zum Kondensator um 180° versetzt, womit der Strom der Spannung um 90° nacheilt. Je höher die Signalfrequenz ω desto höher wird auch der Blindwiderstand. Das Wechselstromverhalten der Grundschaltelemente bei einem sinusförmigen Signal, ist in Abbildung 5.2 illustriert.

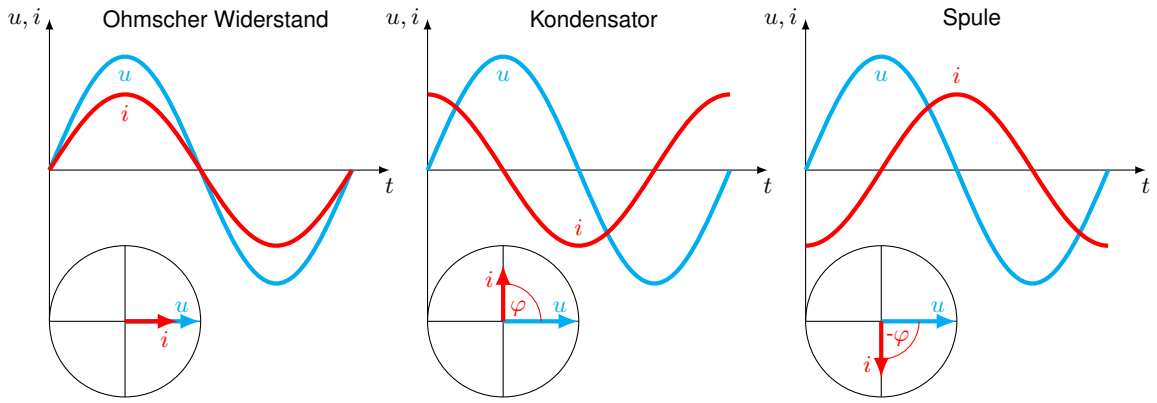

Abbildung 5.2: Phasenverschiebung der Grundschaltelemente Widerstand, Kondensator und Spule.

5.2 Ablauf der Impedanzspektroskopie und Auswertung der Messdaten

Grundsätzlich wird bei der Impedanzspektroskopie ein Wechselstrom- oder Wechselspannungssignal in ein Bauteil oder eine Baugruppe eingebracht und dessen Spannungs- sowie die Stromantwort gemessen. Der Betriebspunkt an dem gemessen wird, kann entweder potentiostatisch (konstante Spannung) oder galvanostatisch (konstanter Strom) sein [65]. Um ein Spektrum zu erhalten, werden verschiedene Frequenzen für eine bestimmte Messdauer eingeprägt, wie Abbildung 5.3 zeigt. Ein typischer Frequenzbereich ist hierbei 1 mHz bis 100 kHz. Wird die Messung während des Betriebs durchgeführt, ist das Einprägesignal um den Gleichstrom- beziehungsweise Gleichspannungsanteil versetzt. Aus diesen Einzelmessungen ergibt sich dann ein Frequenzspektrum mit einer charakteristischen Impedanzkurve.

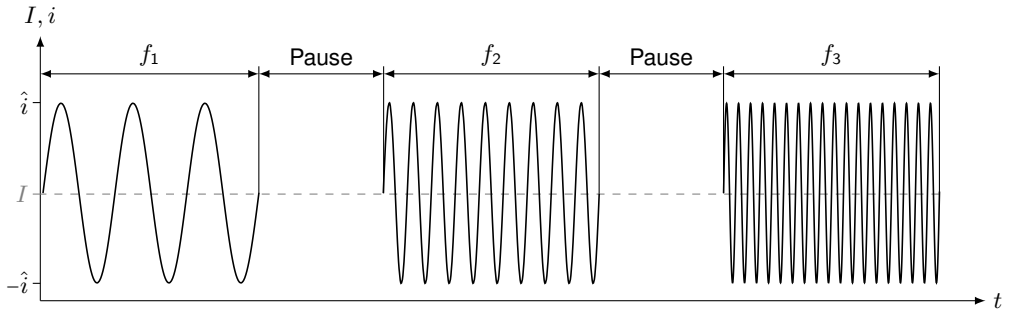

Abbildung 5.3: Abfolge der Frequenzeinprägung bei einer EIS Messung, wobei das Wechselstromsignal um den Betrag des Gleichstromanteils I versetzt ist.

Für eine Messung der Impedanz sind mehrere Prozessschritte notwendig, welche in Abbildung 5.4 aufgelistet sind. Die Schritte 1. bis 3. sind in den nachfolgenden Kapiteln 6 näher ausgeführt. Die Umsetzung des Speicherns und der Darstellung der Ergebnisse wird in Kapitel 8 dargelegt. Nach dem Erfassen der

[65] Vgl. Peter Kurzweil 2020, S. 102.

Messdaten und deren Konvertierung in ein gängiges Dateiformat, erfolgt die Signalanalyse mittel FFT. Diese liefert für jede Frequenz des gemessenen Spektrums einen komplexen Widerstandswert. Aus diesen Werten und den Messfrequenzen lassen sich alle üblichen Auswertungen der Impedanzspektroskopie erzeugen.

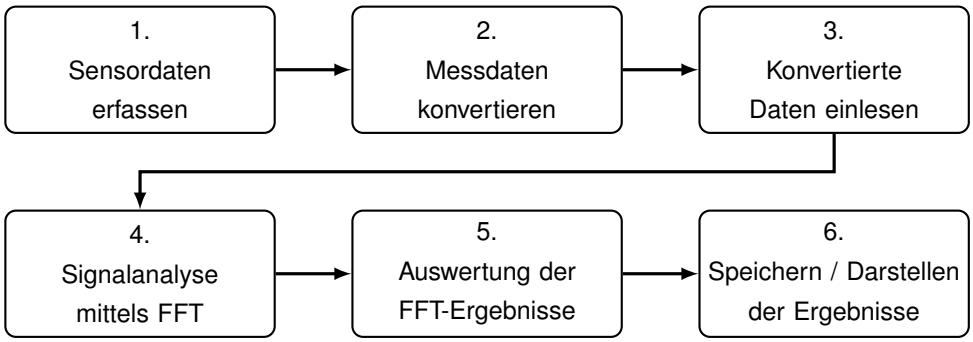

Abbildung 5.4: Ablauf einer Impedanzspektroskopie mit den einzelnen Prozessschritten.

5.2.1 Amplituden und Phasen Darstellung

Bei dieser Darstellung der Daten wird aus dem Real- und Imaginärteil einmal die Amplitude sowie die Phasenverschiebung errechnet und logarithmisch über die Frequenz aufgetragen, wie Abbildung 5.5 exemplarisch zeigt. Um eine Vergleichbarkeit der Daten zu erreichen, werden die Widerstandswerte mit der Zellfläche normiert.

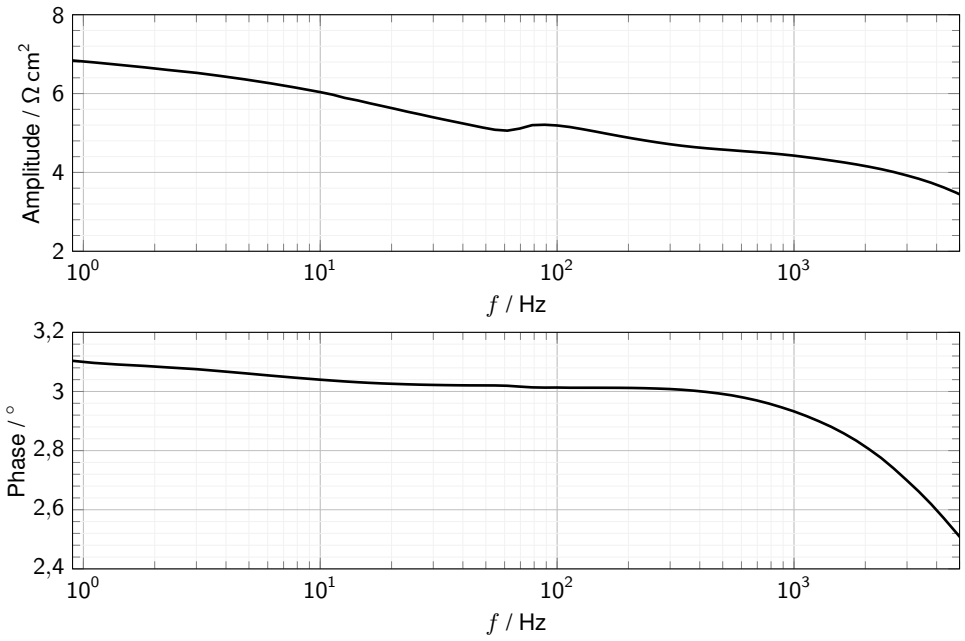

Abbildung 5.5: Amplituden und Phasen Darstellung über logarithmische Frequenz eines PEM-Elektrolyse-Stack.

Diese Art der Darstellung wird als Bode Diagramm bezeichnet. Sie stellt eine einfache Variante dar, um ein lineares und zeitinvarinates System zu beschreiben. Der Vorteil liegt darin, dass der Frequenzgang ersichtlich ist und damit alle Informationen vorliegen. Ändert sich das fundamentale Verhalten des Systems nicht, reagiert auch die Bode Darstellung nicht sehr sensitiv auf Änderungen, was als Nachteil anzusehen ist. [66]

[66] Vgl. PalmSens Knowledge Base 2022.

5.2.2 Ortskurve der Impedanz

Bei der Ortskurve der Impedanz, bekannt als Nyquist Diagramm, wird der Imaginärteil über den Realteil des komplexen FFT-Ergebnisses aufgetragen. Um aus der Kurvenform Rückschlüsse über den Prüfling ziehen zu können, müssen die beiden Achsen proportional zueinander sein. Da dieses Diagramm normalerweise keine Information über die Frequenz enthält, kann diese nachträglich noch einzelnen Punkten zugeordnet werden, wie in Abbildung 5.6 zu sehen. Gegen die übliche Intuition befinden sich die Werte mit der höchsten Frequenz nicht rechts im Diagramm sondern links. Außerdem ist es in der Elektrochemie üblich, die Ordinate im negative Wertebereich darzustellen. Dadurch wird der vierte Quadrant mittels Vorzeichen Wechsel in der Achsenbeschriftung in den ersten Quadranten transferiert.

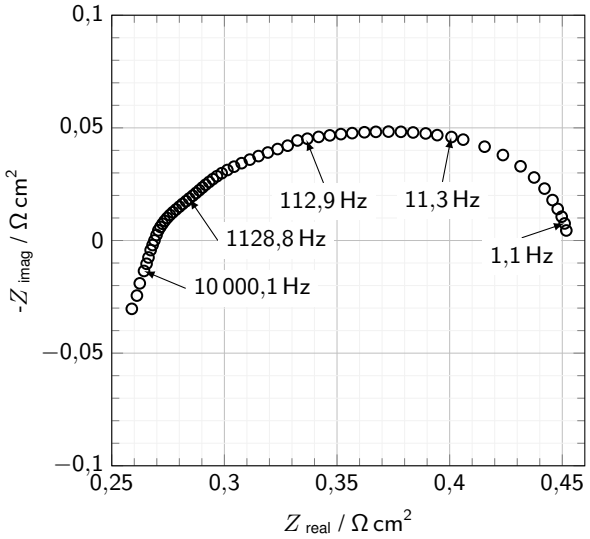

Abbildung 5.6: Ortskurve mit Imaginär- über Realteil einer PEM-Elektrolysezelle.

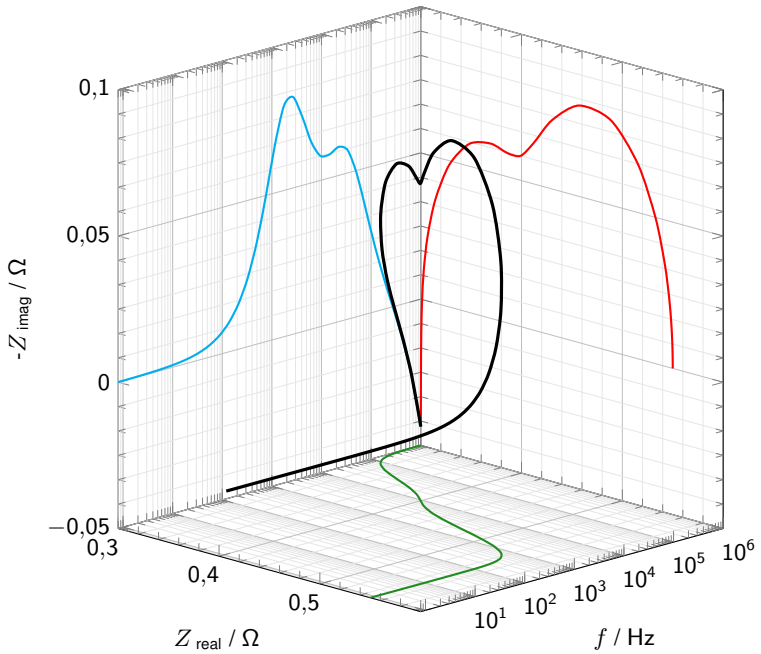

Abbildung 5.7: Ortskurve im dreidimensionalen Raum mit Real und Imaginärteil sowie der Messfrequenz eines fiktiven Ersatzschaltbildes.

Um die Frequenzen sichtbar zu machen, gibt es noch die Möglichkeit das Diagramm im dreidimensionalen Raum darzustellen, wie Abbildung 5.7 zeigt. Über die Projektion in der komplexen Ebene (rot) ist wieder die Ortskurve bestehend aus Real- und Imaginärteil ersichtlich. Auf den beiden anderen Ebenen ist jeweils der Real- (grün) sowie der Imaginärteil (blau) über die Frequenz aufgetragen.

5.2.3 Dynamic-Time-Warping

Eine Methode, um die Form der Ortskurve numerisch zu analysieren, ist das Dynamic-Time-Warping. Ursprünglich kommt diese Methode aus der Spracherkennung und wird immer mehr auch in der Auswertung großer Datensätze verwendet. Ziel ist es, Ähnlichkeiten zwischen Kurven zu erkennen und diese einem numerischen Wert zuzuordnen [67]. Dabei beruht der Algorithmus auf dem einfachen Konzept, den Real- und Imaginärteil der Datenpunkte von zwei Datensätzen zu berechnen, wie Abbildung 5.8 illustriert.

Erfolgt dies für alle Datenpunkte, erhält man zwei Diagramme, die jeweils den errechneten Wert über die Frequenz ausgeben. In diesem fiktiven Beispiel lässt sich erkennen, dass sowohl im Real- als auch im Imaginärteil die größte Veränderung zwischen 100 Hz und 1000 Hz auftritt. Je näher die Werte an der Nulllinie liegen, desto ähnlicher sind sich die beiden Datensätze. Somit ist diese Analysemethode besonders für den Vergleich verschiedener Messungen beziehungsweise unterschiedlicher Versuchsträger geeignet. Liegen Daten über einen langen Zeitraum vor, kann auch die Veränderung einzelner Punkt über die Zeit aufgetragen werden. Durch Bilden der Summe über alle Punkte, ist es möglich einen Indikator für die zeitliche Veränderung des Gesamtsystems zu erhalten.

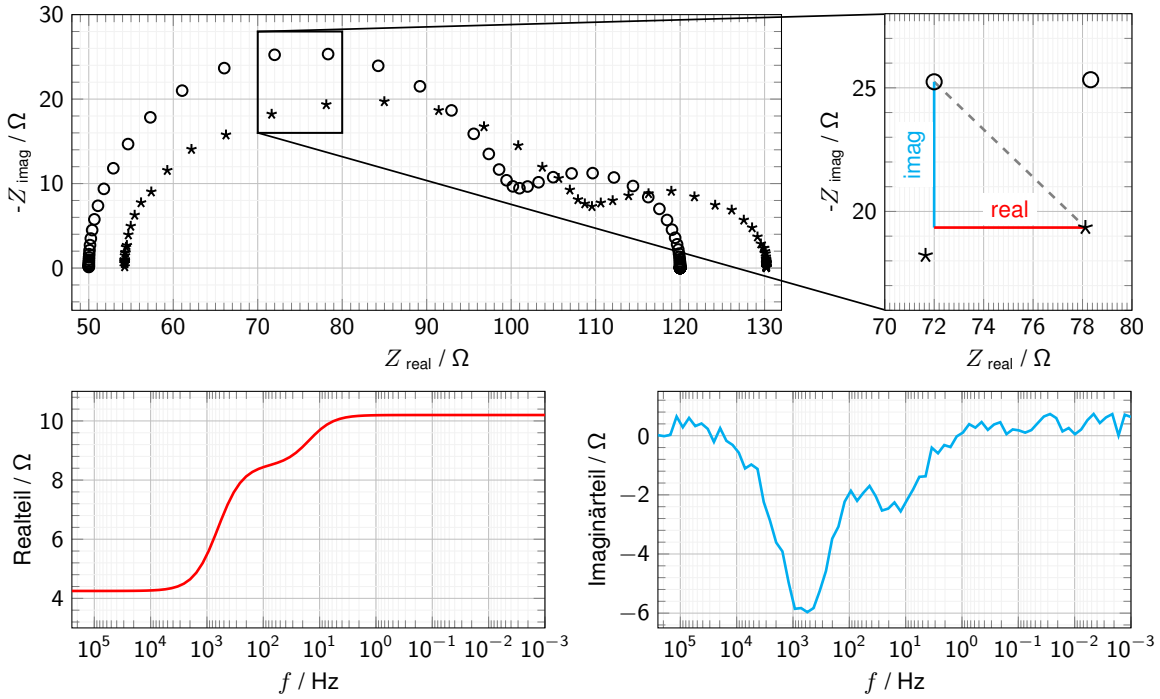

Abbildung 5.8: Berechnung der Ähnlichkeit zweier Datensätze mittels Dynamic-Time-Warping. In den Diagrammen für den Real- und Imaginärteil sind die Frequenz von hoch links zu niedrig rechts aufgetragen analog zur Ortskurve, um die Zuordnung zu erleichtern.

[67] Vgl. M. Durand, A. Picot, J. Rénier, C. Turbin, O. Crassous, M. Scohy, R. Stephan, O. Abassie and C. Andrieux 2021, S. 303–306.

5.3 Elektrisches Ersatzschaltbild einer Impedanzmessung

Die Komponenten eines elektrochemischen Systems können durch elektrische Bauteile in ihrem Frequenzverhalten angenähert werden. Dadurch ist es möglich, Veränderungen in der Ortskurve mit der Änderung der Parameter von elektrischen Bauteilen zu beschreiben, und in weiterer Folge Komponenten von Elektrolysezellen zuzuordnen. Das Zurückführen auf Komponenten hängt jedoch stark von der Güte des Ersatzschaltbildes ab, da durch Rekombination der Bauteile auch Mehrdeutigkeiten entstehen können.

Um das Frequenzverhalten elektrischer Schaltungen nachvollziehen zu können, wird zunächst das Verhalten der Grundbausteine betrachtet und ist in Abbildung 5.9 dargestellt.

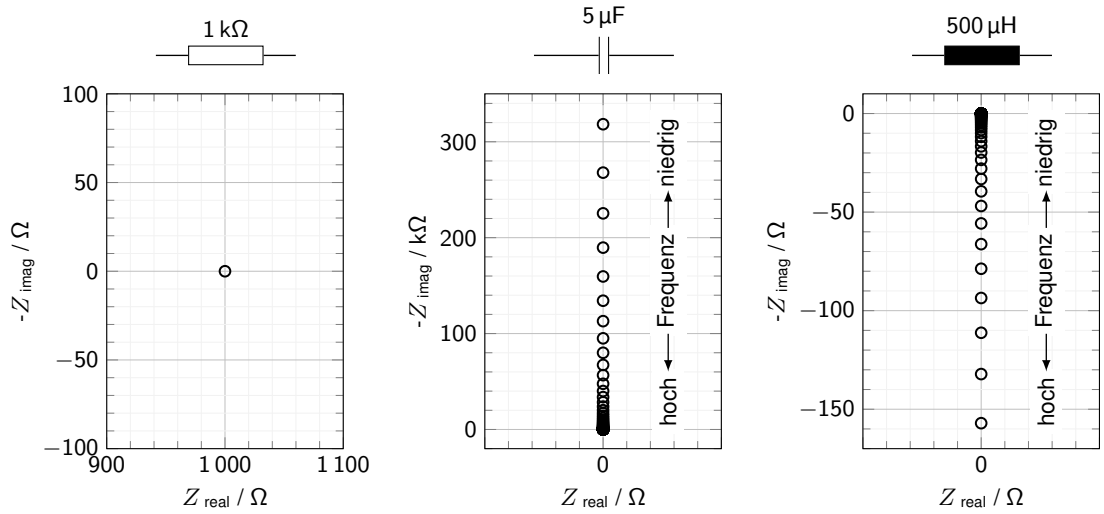

Abbildung 5.9: Frequenzgang von Widerstand, Kondensator und Spule im Nyquist Diagramm, in einem Frequenzbereich von 0.1 Hz bis 50 kHz.

Da sich die Ergebnisse auf einer Linie ausbilden, wird zur besseren Darstellung auf die Proportionalität des Achsenverhältnisses in diesem Fall verzichtet. Wie schon in Abschnitt 5.1 erläutert, erzeugt ein ohmscher Widerstand keine Phasenverschiebung, was sich auch in seinem Frequenzverhalten widerspiegelt. Der Widerstandswert bleib über das hier betrachtete Frequenzband konstant, wobei der Wert dem auf der realen Achse entspricht, welcher wiederum der Bauteilparameter ist. Hierbei lässt sich auch erkennen, warum beim Nulldurchgang ein reiner ohmscher Widerstand in einer Schaltung vorliegt. Im Gegensatz dazu zeigt der Kondensator ein sehr ausgeprägtes Frequenzverhalten, der frequenzabhängige Imaginärteil der Impedanz reicht von 0 Ω bei 50 kHz bis 300 kΩ bei 0,1 Hz. Die Werte sind als exemplarisch zu betrachten, da diese nur für den angegebenen Frequenzbereich und einen Kondensator mit 5 µF gelten. Wie Formel (5.4) zeigt, ergibt sich das lineare Verhalten aus der Proportionalität von Bauteilparameter und Frequenz. Da die Werte im Frequenzband jedoch logarithmisch zunehmen, ergibt sich auch im Diagramm eine nichtlineare Zunahme des Widerstandswertes. Dieser bildet sich nur entlang der imaginären Achse aus, da sich die Impedanz eines Kondensators ausschließlich aus dessen Blindwiderstand ergibt. Die Werte werden mit positivem Vorzeichen aufgetragen, weil die Ordinate negativ skaliert ist. Weil der Blindwiderstand für einen Kondensator immer < 0 ist, ergibt sich auch ein negativer Phasenwinkel

$$\underline{Z}_C = 0 + \mathrm{j}\, X_C \qquad \text{mit} \qquad X_C = -\frac{1}{\omega\, C} < 0, \tag{5.6}$$

$$\varphi = \arctan\left(\frac{\mathrm{Im}(\underline{Z}_C)}{\mathrm{Re}(\underline{Z}_C)}\right) \Longrightarrow \arctan(-\infty) = -\frac{\pi}{2}. \tag{5.7}$$

Die Spule zeigt dasselbe Verhalten nur in umgekehrter Form.

$$\underline{Z}_L = 0 + \mathrm{j}\, X_L \qquad \text{mit} \qquad X_L = \omega\, L > 0, \tag{5.8}$$

$$\varphi = \arctan\left(\frac{\mathrm{Im}(\underline{Z}_L)}{\mathrm{Re}(\underline{Z}_L)}\right) \Longrightarrow \arctan(\infty) = \frac{\pi}{2}. \tag{5.9}$$

Diese erreicht bei niedrigen Frequenzen sehr geringe Widerstandswerte und mit Zunahme der Frequenz steigt auch der Blindwiderstand. Das unterschiedliche Verhalten wird im Vergleich der beiden Bauteile im Bodeplot deutlich, wie Abbildung 5.10 illustriert.

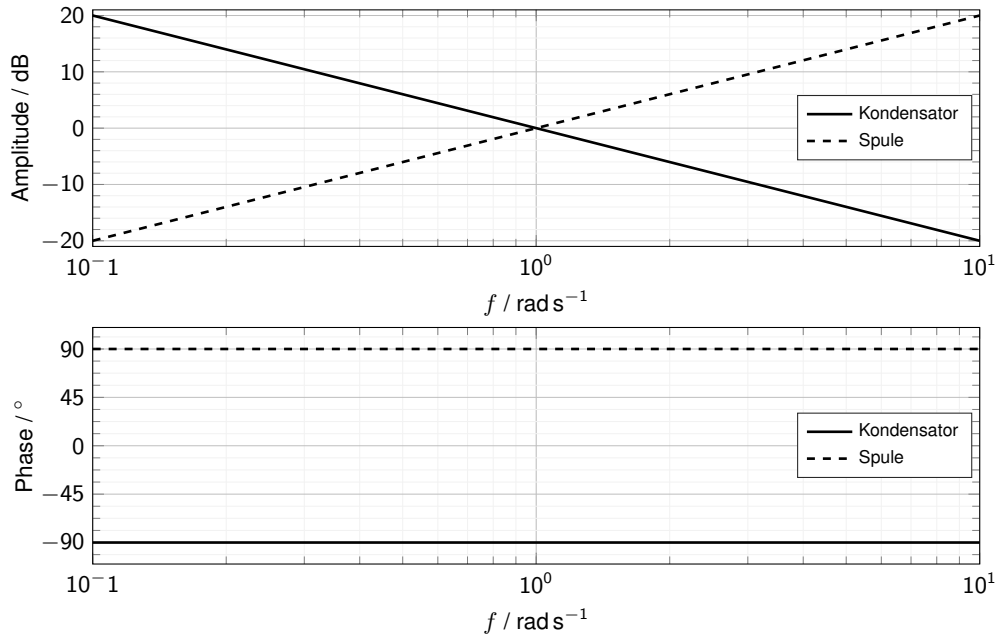

Abbildung 5.10: Amplituden- und Phasengang von Kondensator und Spule.

Auch hier besteht wieder ein proportionaler Zusammenhang mit der Frequenz. Erst wenn man die Grundbausteine zu einem Schaltkreis zusammensetzt, ergeben sich die charakteristischen Ortskurven. In Abbildung 5.11 ist die eines RLC-Gliedes ersichtlich.

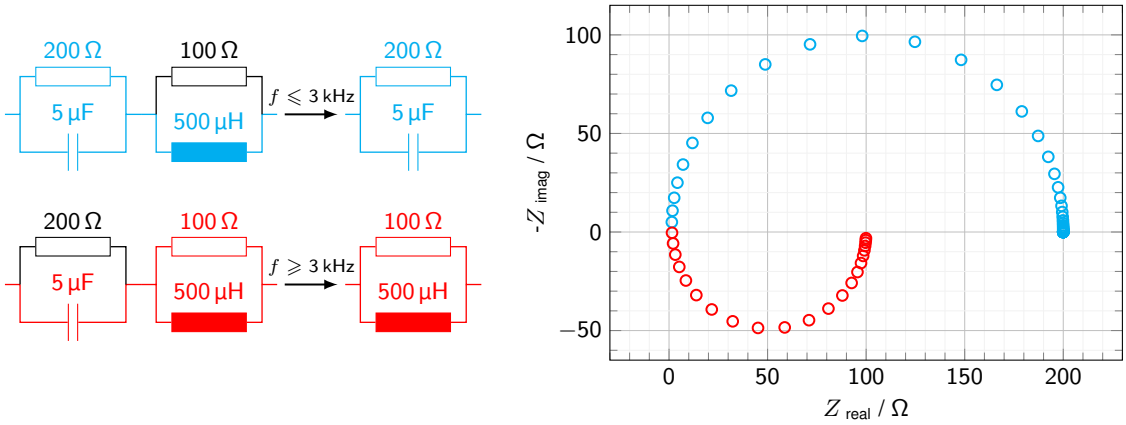

Abbildung 5.11: Frequenzgang von Widerstand, Kondensator und Spule in Parallelschaltung bei einem Frequenzbereich von 0,1 Hz bis 1 MHz.

Dabei besteht die Schaltung aus einem Kondensator und einer Spule, die jeweils parallel mit einem Wi-

derstand verschaltet sind. Der Zusammenhang zwischen Ortskurve und Schaltung lässt sich am besten anhand der beiden Nulldurchgänge an der Abszisse erläutern. Ist die Frequenz niedrig, geht auch die Spulenimpedanz gegen Null, weshalb der parallel geschaltete Widerstand mit 100 Ω keinen Einfluss auf die Kurve hat. Berechnet man für die Frequenz von $f = 0{,}1\,\mathrm{Hz}$ die Impedanz der verbliebenen Parallelschaltung aus Widerstand und Kondensator, so ergibt sich ein Wert von

$$|\underline{Z}| = \frac{1}{\sqrt{\left(\frac{1}{R_C}\right)^2 + (\omega\, C)^2}} = \frac{1}{\sqrt{\left(\frac{1}{200\,\Omega}\right)^2 + (2\,\pi \cdot 0{,}1\,\mathrm{Hz} \cdot 5\,\mu\mathrm{F})^2}} \approx 200\,\Omega, \tag{5.10}$$

$$\text{mit} \qquad \varphi = \arctan\left(-\omega\, C R_C\right) = \arctan\left(-2\,\pi \cdot 0{,}1\,\mathrm{Hz} \cdot 5\,\mu\mathrm{F} \cdot 200\,\Omega\right) \approx 0^\circ. \tag{5.11}$$

Aus dem Winkel und der Impedanz lässt sich der Realteil des Widerstandes berechnen

$$\mathrm{Re}(\underline{Z}) = \underline{Z}\,\cos\left(\varphi\right) = \cos\left(0^\circ\right) \cdot 200\,\Omega = 200\,\Omega. \tag{5.12}$$

Dieser Wert entspricht mit gerundeten Werten dem im Diagramm bei niedrigen Frequenzen, sowie dem Wert des parallel geschalteten Widerstandes. Bei hoher Frequenz geht der Blindwiderstand des Kondensators gegen Null, weshalb der parallel geschaltete Widerstand mit 200 Ω keinen Einfluss auf die Kurve hat. Analog kann auch hier wieder der Scheinwiderstand berechnet werden

$$|\underline{Z}| = \frac{1}{\sqrt{\left(\frac{1}{R_L}\right)^2 + \left(\frac{1}{\omega\, L}\right)^2}} = \frac{1}{\sqrt{\left(\frac{1}{100\,\Omega}\right)^2 + \left(\frac{1}{2\,\pi \cdot 1\,\mathrm{MHz} \cdot 500\,\mu\mathrm{H}}\right)^2}} \approx 100\,\Omega, \tag{5.13}$$

$$\text{mit} \qquad \varphi = \arctan\left(\frac{R_L}{\omega\, L}\right) = \arctan\left(\frac{100\,\Omega}{2\,\pi \cdot 1\,\mathrm{MHz} \cdot 500\,\mu\mathrm{H}}\right) = 1{,}82^\circ. \tag{5.14}$$

Womit sich der Real- und Imaginärteil des Widerstandes wie folgt ergibt

$$\mathrm{Re}(\underline{Z}) = \underline{Z}\,\cos\left(\varphi\right) = \cos\left(1{,}82^\circ\right) \cdot 157\,\Omega \approx 100\,\Omega, \tag{5.15}$$

$$\mathrm{Im}(\underline{Z}) = \underline{Z}\,\sin\left(\varphi\right) = \sin\left(1{,}82^\circ\right) \cdot 157\,\Omega \approx 3\,\Omega. \tag{5.16}$$

Auch diese berechneten Werte stimmten mit dem letzten Datenpunkt im Diagramm 5.11 überein. Somit dominiert der kapazitive Anteil der Schaltung den niederfrequenten Bereich und der induktive Anteil den hochfrequenten. Die Induktivität erkennt man auch daran, dass die Ortskurve bis in den negativen Bereich der gespiegelten Ordinate reicht. Bei einer rein kapazitiven Schaltung wird dieser negative Bereich nie erreicht, wie Abbildung 5.16 zeigt.

Constant-Phase-Element

Da sich nicht alle elektrochemischen Prozesse mit elektrischen Bauteilen abbilden lassen, müssen neue Elemente eingeführt werden. Eines davon ist das Constant-Phase-Element (CPE), dessen Eigenschaften und Schaltzeichen dem eines Kondensators ähnelt und mathematisch über

$$Z_Q = \frac{1}{Q_0\,(\mathrm{j}\,\omega)^n} \tag{5.17}$$

beschrieben ist, wobei das Element mit Q_0 und n als charakteristischen Parameter beschrieben wird. Q_0 hat die Einheit $\mathrm{S\,s^n}$ (Siemens Sekunde hoch n), was wiederum keiner realen physikalische Bedeutung zugeordnet werden kann. Die Umrechnung vom CPE auf eine Kapazität ist nur für $n > 0{,}75$ und nur für

eine Parallelschaltung mit einem Widerstand gültig. Der Zusammenhang ergibt sich über die Zeitkonstante τ wie folgt [68]

$$RQ_0 = \tau^n = (RC)^n \qquad \Longrightarrow \qquad C = \frac{(RQ_0)^{\frac{1}{n}}}{R}. \tag{5.18}$$

Das CPE beschreibt Kapazität, wie sie in elektrochemischen Systemen auftritt. Dabei besitzt dieses auch einen Realanteil, wie der Frequenzgang bei unterschiedlichen Exponentenwerten in Abbildung 5.12 illustriert. Der Zusammenhang zwischen Winkel und Exponenten wird mit $n \cdot (-90°)$ beschrieben. In einer Parallelschaltung mit einem Widerstand ergibt sich der für einen Kondensator charakteristische Halbkreis, nur das der Mittelpunkt des Kreises bei Werten $n < 1$ unterhalb der Nulllinie der Abszisse liegt, wie das zweite Diagramm in Abbildung 5.12 unterstreicht.

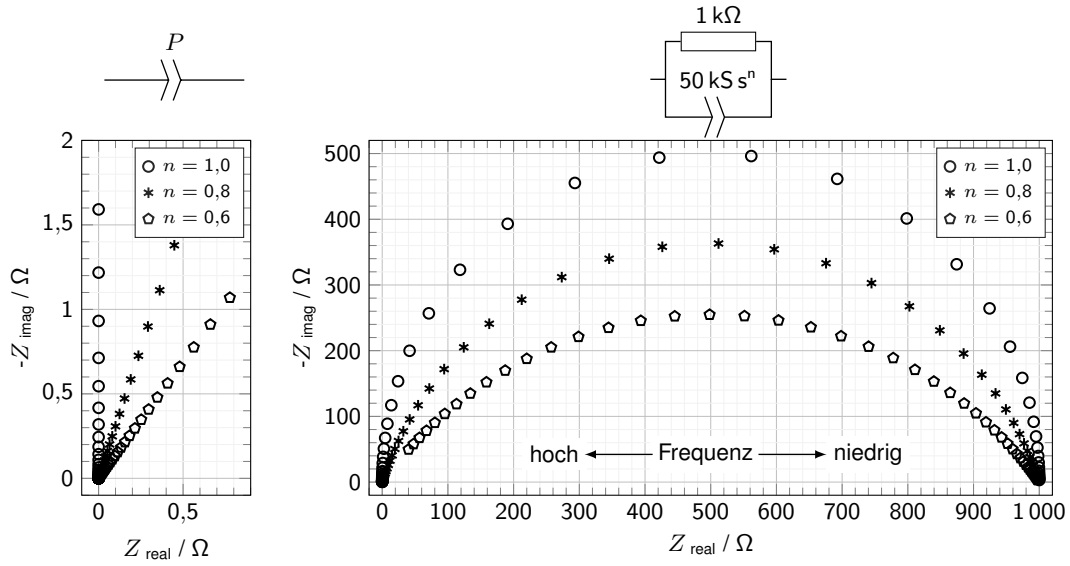

Abbildung 5.12: Frequenzgang eines Constant Phase Elements links. Im rechten Diagramm in Parallelschaltung mit einem Widerstand.

Ein Grund warum das CPE benötigt wird, sind raue Oberflächen der Elektroden, die sich mit zunehmender Rauigkeit stark vergrößert. Dies spiegelt sich im Exponenten n wider, dessen Wertebereich zwischen 0 und 1 liegt. Raue Oberflächen liegen ungefähr bei einem Wert von $n = 0{,}5$ und gehen, je glatter sie werden, bis zu $n = 1$. Auch inhomogen Reaktionsraten auf der Oberfläche führen dazu, dass die elektrochemischen Vorgänge nicht mehr mit einem elektrischen Kondensator beschrieben werden können. Dazu zählt auch eine nicht konsistente Dicke von Beschichtungen, wie etwa die der Katalysatoren. Ein weiterer Einflussfaktor ist die inhomogene Stromverteilung über die Oberfläche, wie es etwa an den Rändern der Zellen der Fall ist. Diese Faktoren können einzeln oder im Zusammenspiel wirken, was wiederum eine genaue Zuordnung der Ursache erschwert. [69] [70]

Der Einfluss des Constant-Phase-Elements ist in Abbildung 5.13 mit drei fiktiven Schaltbildern ersichtlich. Zum besseren Vergleich ist in Schaltbild 1 ein RC-Glied definiert. Schaltbild 2 zeigt im hochfrequenten Bereich besonders beim Annäherungswinkel hin zum Hochfrequenzwiderstand eine deutliche Winkeländerung. Aufgrund des geringeren Parameterwertes im Vergleich zum Kondensator, nähert sich die Kurve im niederfrequenten Bereich wieder dem ersten Schaltbild an. Im Vergleich zum Dritten wird der Einfluss eines höheren Parameterwertes deutlich, da hier auch der niederfrequente Bereich beeinflusst wird.

[68] Vgl. Research Solutions & Resources LLC 2022.
[69] Vgl. Jean-Baptiste Jorcin and Mark E. Orazem and Nadine Pébère and Bernard Tribollet 2006, S. 1.
[70] Vgl. Matt Lacey 2022b.

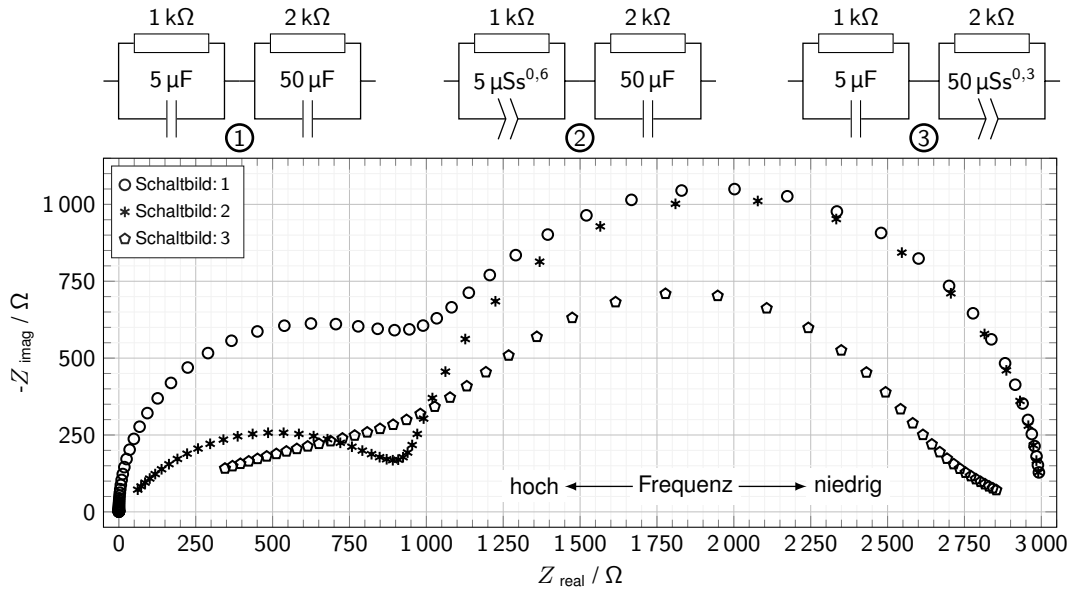

Abbildung 5.13: Vergleich des Frequenzganges verschiedener Schaltbilder bestehend aus Widerstand, Kondensator und Constant-Phase-Element.

Warburg-Element

Die gebräuchlichste Methode um ein Diffusionsverhalten zu modellieren ist das Warburg-Element. Dieses beschreibt eine halbunendliche lineare Diffusion, was einer uneingeschränkten Diffusion an einer großen und ebenen Elektrode entspricht. Da nur der Abstand zur Elektrode von Bedeutung ist, stellt dies die einfachste Diffusionssituation dar. Die Größe des Elements ist umgekehrt proportional zur Quadratwurzel der Frequenz $\omega^{-1/2}$, wobei der Real- und Imaginärteil immer denselben Wert annehmen

$$\underline{Z}_W = \sigma\,\omega^{-1/2} - \mathrm{j}\,\sigma\,\omega^{-1/2} \qquad \Longrightarrow \qquad |\underline{Z}_W| = \sqrt{2}\,\sigma\,\omega^{-1/2}. \tag{5.19}$$

Der Warburgkoeffizient σ ist der Diffusionskoeffizient von Ionen in Lösung. Wobei R für die ideale Gaskonstante, T für die thermodynamische Temperatur, F für die Faraday Konstante, A beschreibt die Oberfläche und n für die Wertigkeit steht. D steht für den Diffusionskoeffizienten, der noch über seinen Index in Oxidation O und Reduktion R unterschieden wird. Die jeweilige Konzentration wird durch C^b beschrieben [71]

$$\sigma = \frac{\mathrm{R}\,T}{n^2\,F^2\,A\,\sqrt{2}}\left(\frac{1}{D_O^{1/2}\,C_O^b} + \frac{1}{D_R^{1/2}\,C_R^b}\right) \qquad \text{mit} \qquad R = 8{,}314\,462\,618\,153\,24\ \mathrm{J\,mol^{-1}\,K^{-1}}. \tag{5.20}$$

Charakteristisch für das Warburg Element ist damit der konstante Phasenwinkel mit 45°, unabhängig von der Frequenz. Deshalb kann die Warburg-Impedanz auch durch ein Constant-Phase-Element mit n = 0,5 beschrieben werden. Das Frequenzverhalten des Warburg-Elements ist in Abbildung 5.14 dargestellt. Die Größe des Warburgkoeffizienten bestimmt, wie weit sich die Datenpunkte vom Ursprung entfernen. Dabei gilt, je größer der Koeffizient ist, desto weiter entfernen sich die Punkte vom Achsenursprung. Der Abstand steigt aber auch mit sinkender Frequenz und konstantem Koeffizienten an, weshalb das Warburg-Element die Ortskurve vor allem im niederfrequenten Bereich beeinflusst. In einer Parallelschaltung mit einem Widerstand zeigt sich ein kondensatorähnliches Verhalten erst bei hohen Werten für σ. Je nach Betrag des Koeffizienten wird die Ausprägung der Halbkreiskurve beeinflusst.

[71] Vgl. Matt Lacey 2023.

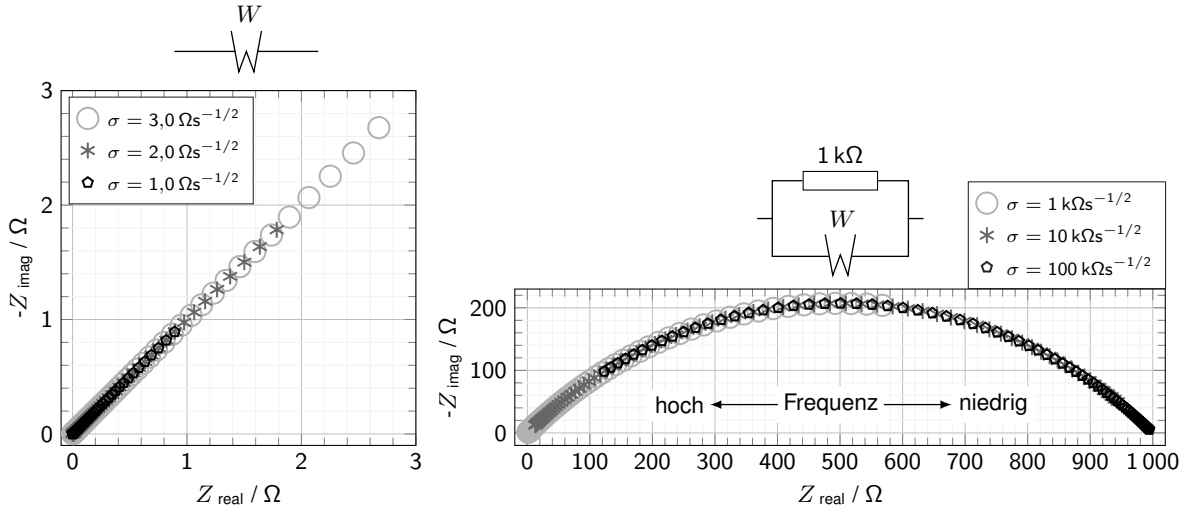

Abbildung 5.14: Frequenzgang eines Warburg-Elements links, rechts im Diagramm in Parallelschaltung mit einem Widerstand.

Der Einfluss des Warburg-Elements auf ein Schaltbild mit RC-Glied ist in Abbildung 5.15 illustriert.

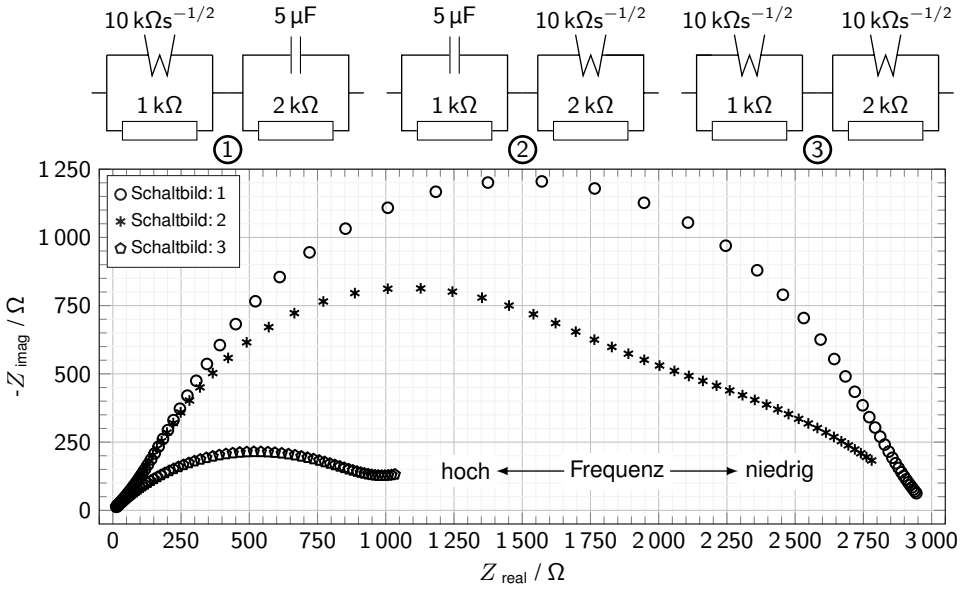

Abbildung 5.15: Vergleich des Frequenzganges verschiedener Schaltbilder bestehend aus Widerstand, Kondensator und Warburg-Element.

Je nach Verhältnis von Kapazität zu Warburgkoeffizienten bilden sich die Anlauf- und Auslaufwinkel unterschiedlich aus und ergeben somit nicht die charakteristischen $45°$. Auch die doppelte Parallelschaltung mit zwei Warburg-Elementen weist diese Charakteristik nicht auf. Es sei noch erwähnt, dass es sich hierbei um rein fiktive Schaltungen sowie Komponentenwerte handelt.

5.3.1 Hoch- und Niederfrequenzwiderstand

Nicht nur aus der Form der Ortskurve kann man Informationen über das System generieren, sondern auch über die Schnittpunkte des Imaginärteils mit der Nulllinie. Ist dieser Null, liegt ein rein ohmscher Widerstand

vor, der sich im Wert des Realteils widerspiegelt. Je nach Art des Systems kann dieses keinen, einen oder mehrere haben. Zur Veranschaulichung wird in Abbildung 5.16 ein Schaltkreis mit einer Frequenz von 1 mHz bis 250 kHz simuliert.

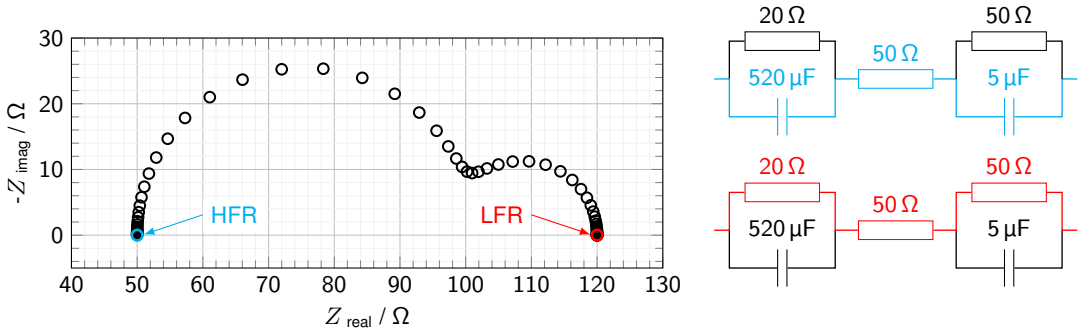

Abbildung 5.16: Hoch- und Niederfrequenzwiderstand einer Schaltung sowie der Stromverlauf bei hoher und niedriger Frequenz.

Die Werte der Komponenten sowie deren Anordnung sind so gewählt, dass sich zwei Nulldurchgänge ausbilden. Wie im vorherigen Abschnitt 5.2.2 beschrieben, liegen die Messwerte bei hoher Frequenz links und bei niedriger rechts im Diagramm. Der jeweilige Nulldurchgang wird deshalb als High Frequency Resistance (HFR) und Low Frequency Resistance (LFR) bezeichnet. Aufgrund des Frequenzverhaltens der Kondensatoren, welches im Bode Diagramm 5.17 dargestellt ist, ändert sich auch der Strompfad durch die Schaltung.

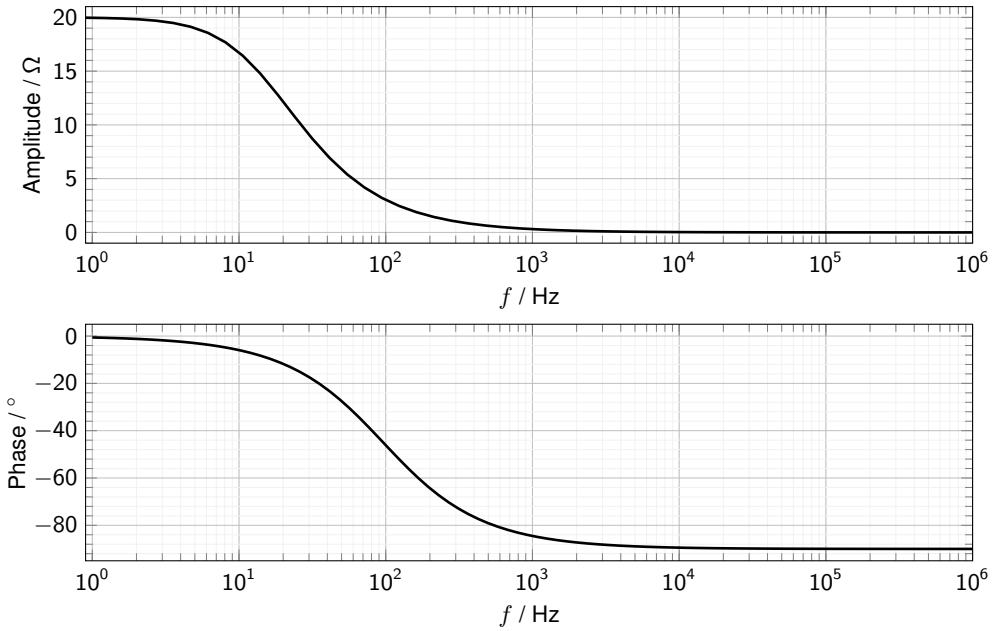

Abbildung 5.17: Bode Diagramm eines parallelen RC-Gliedes mit einem Widerstandswert von 20 Ω sowie einer Kapazität von 520 µF.

Ist die Frequenz hoch ist der Blindwiderstand gering, wodurch die parallelen Widerstände umgangen werden und nur der mittige Widerstand aus Abbildung 5.16 zum Tragen kommt. Ist die Frequenz jedoch niedrig, so zeigt Formel (5.4) dass sich der Blindwiderstand dazu proportional erhöht, weshalb sich der rötliche Pfad über die drei Widerstände ergibt. Im Diagramm finden sich die Beispielwerte der jeweiligen Schaltung wieder. Die 50 Ω des Hochfrequenzpfades entsprechen dem bläulich markierten Nulldurchgang im Diagramm.

Der Pfad bei niedrigen Frequenzen hat diesen Durchgang bei 120 Ω. Dies entspricht der Summe der Widerstände in einer Serienschaltung. Somit ist es möglich, den gesamten ohmschen Widerstand der Schaltung zu ermitteln oder den mittig positionierten Widerstand isoliert zu messen.

Wie Abschnitt 5.3 zeigen wird, können elektrische Schaltungen und deren Komponenten Bauteile eines Elektrolyseurs zugeordnet werden. Somit ist es möglich, die Veränderung von zum Beispiel einer Membran zu dokumentieren. Auch ein Vergleich einzelner Zellen zueinander ist möglich. So kann der Widerstandswert und die Frequenz, bei der sich der Nulldurchgang einstellt, aufgetragen werden, wie Abbildung 5.18 zeigt. Aber auch andere Einflussgrößen, wie etwa verschiedene Stromdichten, werden aufgetragen [72].

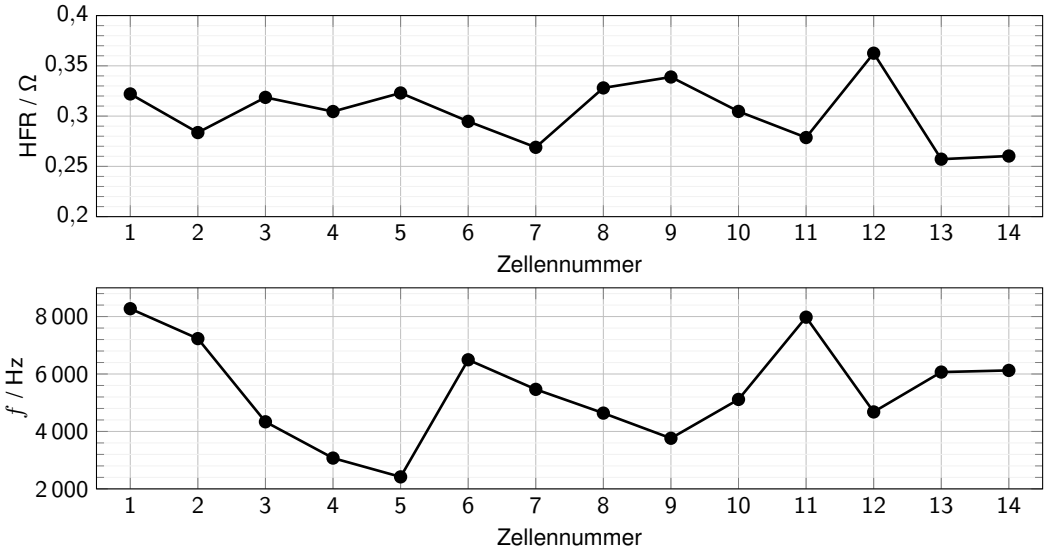

Abbildung 5.18: Widerstand und Frequenz des HFR für 14 Einzelzellen eines PEM-Elektrolyse-Stacks.

5.3.2 Ersatzschaltbild eines PEM-Elektrolyseurs

Nachdem das Verhalten der elektronischen Grundbausteine sowie der Bausteine zur Beschreibung elektrochemischer Vorgänge aufgezeigt wurden, geht es im nächsten Schritt um die Beschreibung eines realen Systems. Hierfür müssen die passenden Bauteile ausgewählt und richtig angeordnet werden. Ein solches Ersatzschaltbild mit der Zuordnung zur jeweiligen Komponente eines PEM-Elektrolyseurs ist in Abbildung 5.19 ersichtlich [73]. Die Ionenleitfähigkeit der Membran kann durch einen ohmschen Widerstand abgebildet werden. Die beiden Elektroden beziehungsweise Katalysatoren sind durch eine Parallelschaltung aus Polarisationswiderstand und Constant-Phase-Element (CPE) beschrieben. Dabei dient das CPE dazu, die Abweichungen von einem idealen kapazitiven Verhalten zu beschreiben. Der Widerstand auf der Kathodenseite entspricht den Verlusten durch die Prozesswärme und auf der Anodenseite kommt zusätzlich die Gibbs-Energie, welche das thermodynamische Potenzial beschreibt, hinzu [74]. Darauf folgt ein Warburg-Element in Serie. Dieses umfasst mögliche Beschränkungen im Massentransport des Wasserstoff- und Sauerstoffgases weg von den Grenzschichten der Elektrode über die Gasdiffusionsschicht. Als letztes Bauteil ist noch ein Widerstand nachgeschaltet, welcher die Grenzflächenimpedanz sowie den metallischen Widerstand repräsentiert [75]. Analog zum symmetrischen Aufbau der Zelle ist auch der gleichwertige Schaltkreis, in Bezug auf die Komponententypen, symmetrisch um den Membranwiderstand aufgebaut.

[72] Vgl. C. Immerz, B. Bensmann, P. Trinke, M. Suermann and R. Hanke-Rauschenbach 2018, F1295.
[73] Vgl. C. Rozain, P. Millet 2014, S. 162.
[74] Vgl. D. Guilbert and G. Vitale 2019, S. 4.
[75] Vgl. C. Rozain, P. Millet 2014, S. 161.

Was in diesem Ersatzschaltbild vernachlässigt wird, ist die über die Zuleitung und parasitären Ströme eingebrachte Induktivität, ohne die es keinen Nulldurchgang der Ortskurve geben würde.

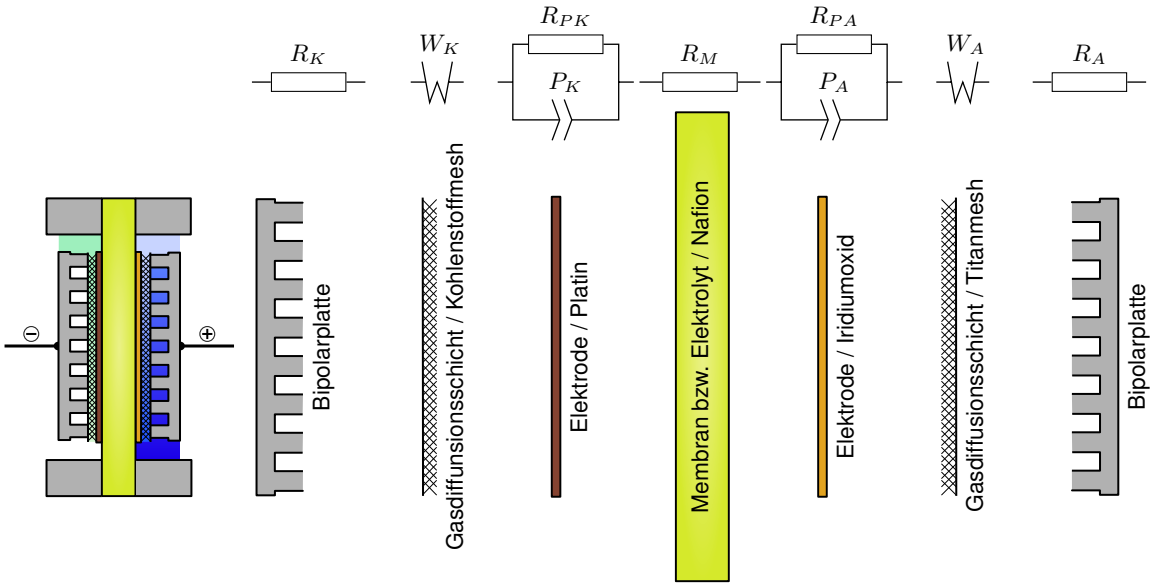

Abbildung 5.19: Ersatzschaltbilder der einzelnen Komponenten einer PEM-Elektrolysezelle mit Widerstand, Constant-Phase-Element und Warburg-Element.

5.3.3 Einfluss der einzelnen Komponenten des PEM-Elektrolyseurs auf die Ortskurve

Es finden sich in der Literatur auch einfachere Schaltbilder mit weniger Komponenten, die nur aus einem Widerstand und zwei RC-Gliedern bestehen [76], wie in Abbildung 5.20 dargestellt.

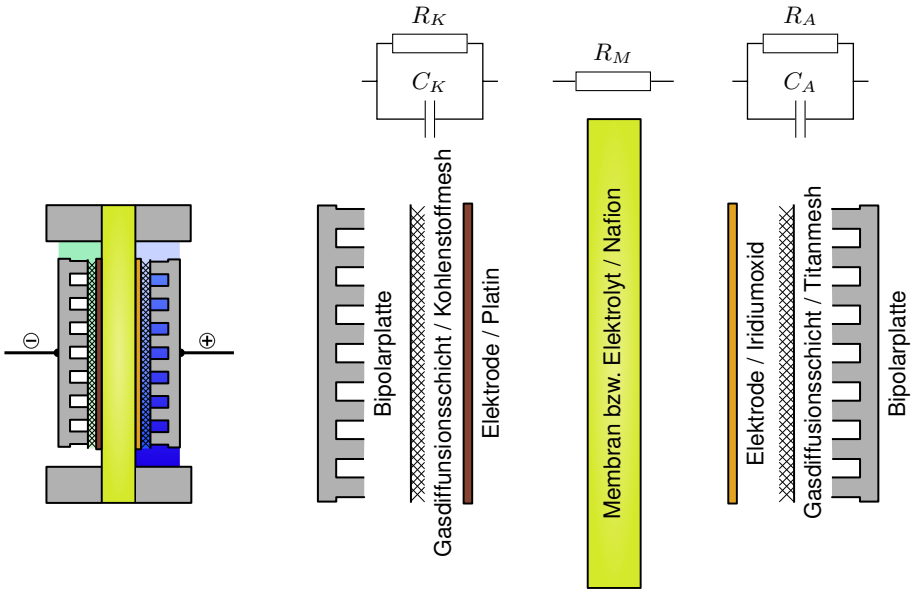

Abbildung 5.20: Vereinfachtes Ersatzschaltbild eines PEM-Elektrolyseurs bestehend aus einem Membranwiderstand und zwei RC-Gliedern die Elektroden und Gasdiffusionsschichten beschreiben.

[76] Vgl. Guilbert, Damien and Vitale, Gianpaolo 2020, S. 4.

Hierbei steht der ohmsche Widerstand R_M für die Membran und die beiden RC-Glieder jeweils für die Elektrode und Gasdiffusionsschicht der Kathode K sowie der Anode A. Um die Veränderung der Ortskurve aufgrund unterschiedlicher Betriebszustände oder Degradation aufzuzeigen, wird ein solches vereinfachtes Schaltbild herangezogen. Durch die Variation bestimmter Komponentenwerte kann deren Einfluss auf die Ortskurve veranschaulicht werden. So zeigt sich in Abbildung 5.21 eine Änderung des Membranwiderstandes von 50 Ω auf 100 Ω in einer Verschiebung der Kurve auf der Abszisse. Wie in Abschnitt 5.3.1 gezeigt, kann über den Hochfrequenzwiderstand der Membranzustand isoliert analysiert werden, da die Nulldurchgänge in der linken Bildhälfte den Widerstandswerten des Membranwiderstandes entsprechen. Die Verschiebung in der Ortskurve entspricht der Differenz der beiden Widerstandswerte, wobei die Membran keinen Einfluss auf die Form der Ortskurve hat. Damit gilt, je geringer die Ionenleitfähigkeit des Elektrolyten in den Elektroden und in der Membran ist, desto höher der ohmsche Widerstand. Die Differenz aus dem Hoch- und Niederfrequenzwiderstand entspricht dem ohmschen Widerstand ohne Membran.

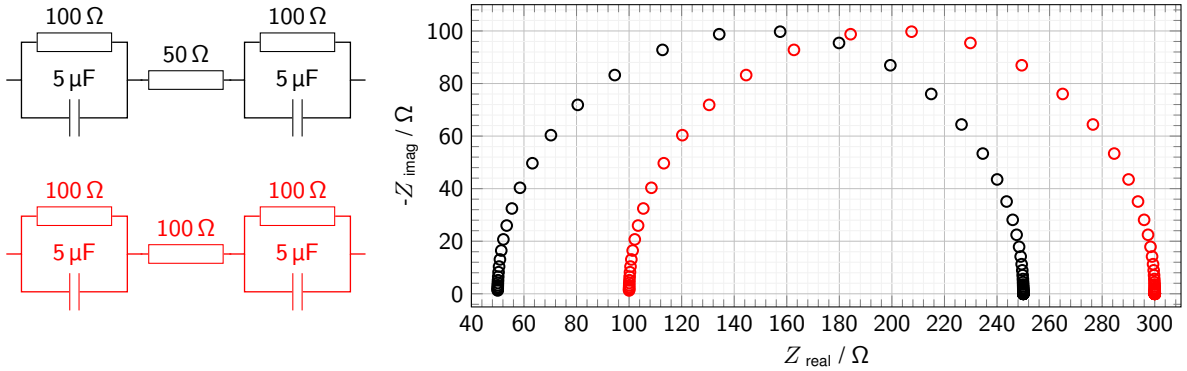

Abbildung 5.21: Ortskurve mit variiertem Membranwiderstand.

Der Einfluss der Kapazität auf die Ortskurve ist in Abbildung 5.22 ersichtlich.

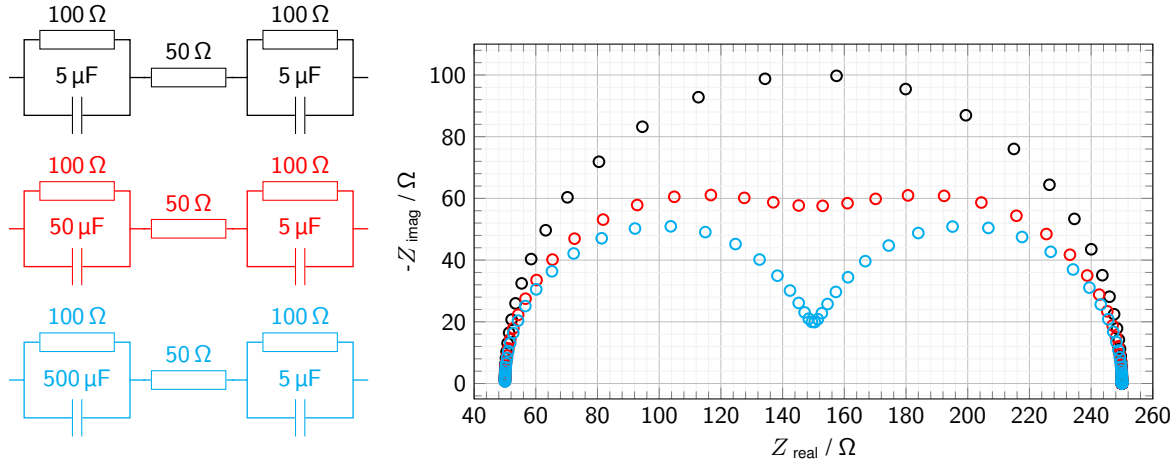

Abbildung 5.22: Ortskurve mit steigender Differenz der Kapazität, welche verantwortlich für das Ausbilden der zwei charakteristischen Halbkreise sind.

Solange die beiden Kapazitätswerte ident sind, bildet sich nur ein Halbkreis aus, wie das schwarze Schaltbild verdeutlicht. So sorgen sehr kleine Werte wie etwa 5 nF dafür, dass sich der Halbkreis nur bei niedrigen Frequenzen und damit in der rechten Hälfte ausbildet. Bei sehr hohen Werten bildet sich dieser nur im linken und damit hochfrequenten Bereich aus. Damit sich die für elektrochemische Prozesse charakteristischen

zwei Halbkreise ausbilden, muss eine Differenz in den Kapazitätswerten vorliegen, wie der rote und blaue Schaltkreis verdeutlichen. Je größer dieser Unterschied ist, umso deutlicher kommen die beiden Halbkreise zum Vorschein.

Über die Widerstände in der Parallelschaltung wird der Durchmesser dieser Halbkreise definiert. Diese lassen sich in Abbildung 5.23 auf der Ordinate ablesen und dem jeweiligen RC-Glied im Schaltbild zuordnen. Somit wird die Form maßgebend von den Widerstandswerte diktiert und damit von der Prozesstemperatur, welche die spezifischen Materialwiderstände beeinflusst, und den Diffusionsverlusten. Diese Verluste ergeben sich aufgrund der Bildung von Gasblasen, die wiederum die Oberfläche auf den Katalysatoren verringern und so den Widerstand erhöhen. Je mehr Energie in Form von Wärme verloren geht, desto größer werden die Durchmesser der charakteristischen Halbkreise. Da der Membranwiderstand bei allen Schaltbildern ident ist, ergibt sich auch bei allen derselbe hochfrequente Nulldurchgang bei $50\,\Omega$.

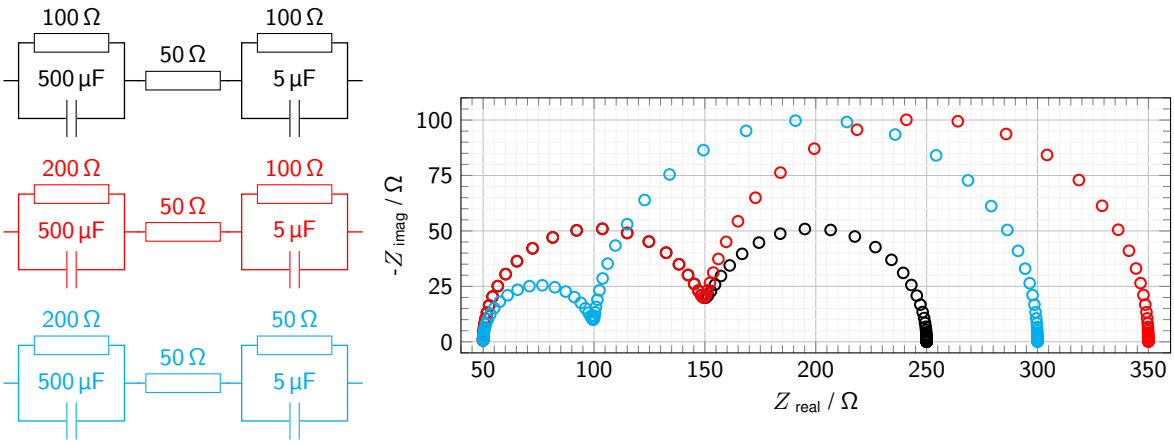

Abbildung 5.23: Ortskurve mit unterschiedlichen Widerstandswerten und Kapazitäten.

Der niederfrequente Nulldurchgang ergibt sich aus allen ohmschen Widerständen und lässt sich anhand der Summe der Widerstandswerte ermitteln. In Abbildung 5.24 sind die einzelnen Schaltungsteile der Ortskurve zugeordnet. Die sich bildenden Halbkreise werden üblicherweise mit Ladungstransporten in Verbindung gebracht. Der Ladungstransport passiert an der Oberfläche der Elektrode, wodurch sich ein Doppelschichtkondensator bildet der geladen wird, was in einer halbkreisförmigen Ortskurve resultiert. In Abbildung 5.24 sind zusätzlich noch die beiden höchsten Punkte (Grenzfrequenz) des jeweiligen Halbkreises mit f_1^* und f_2^* markiert.

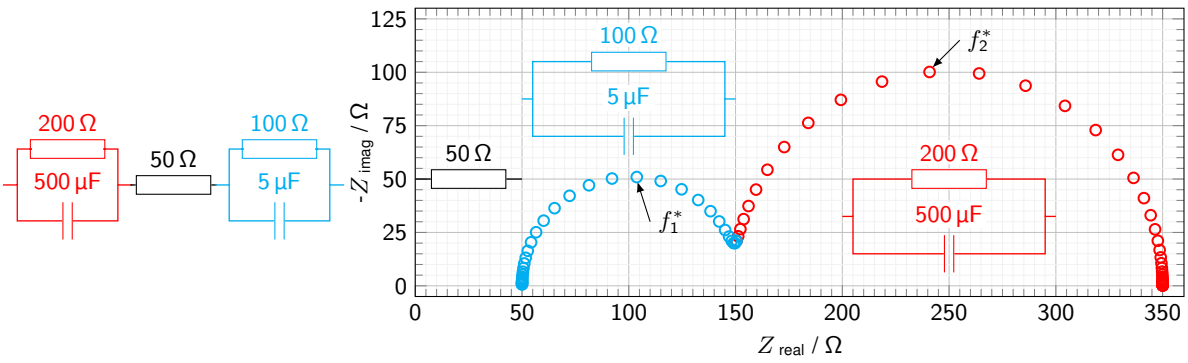

Abbildung 5.24: Ortskurve mit Zuordnung zu Schaltungsteilen und eingezeichneten Grenzfrequenzen am höchsten Punkt der jeweiligen Halbkreise.

Hierbei handelt es sich um die beiden Grenzfrequenzen, die im Englischen und vor allem in der Elektrochemie auch als relaxation frequency bezeichnet wird [77]. Diese Frequenzen korrelieren mit der Zeitkonstante τ eines seriellen RC-Gliedes

$$\tau = \frac{1}{\omega} = \frac{1}{2\,\pi f} = R\,C. \tag{5.21}$$

Daraus ergibt sich die Bedingung für den Maximalwert des Halbkreises mit

$$\omega\,R\,C \stackrel{!}{=} 1. \tag{5.22}$$

Mit der Berechnung der Zeitkonstante τ kann eine Aussage über die Zeitskala, in der die elektrochemischen Prozesse ablaufen, getroffen werden. Dies ist vor allem deshalb wichtig, weil die beiden Halbkreise der Anode beziehungsweise Kathode zugeordnet werden müssen. Nur über das Schaltbild kann diese Zuordnung nicht erfolgen, da dieses das gleiche Ergebnisse liefert, wenn die RC-Glieder vertauscht werden. Somit stellt die Zeitkonstante die Verbindung zur Wasserstoff- und Sauerstoffreaktion in einer PEM-Elektrolysezelle her. Die Grenzfrequenzen lassen sich auch aus einem Graphen des Imaginärteils über die Frequenz ablesen, wie dies in Abbildung 5.25 der Fall ist.

Auch rechnerisch lassen sich die beiden Werte bestimmen über

$$f_1^* = \frac{1}{2\,\pi\,R_K\,C_K} = \frac{1}{2\,\pi \cdot 200\,\Omega \cdot 500\,\mu\text{F}} = 1{,}59\,\text{Hz},$$
$$f_2^* = \frac{1}{2\,\pi\,R_A\,C_A} = \frac{1}{2\,\pi \cdot 100\,\Omega \cdot 5\,\mu\text{F}} = 318{,}31\,\text{Hz}.$$

sofern der Widerstand und die Kapazität bekannt sind. Diese Frequenzwerte sind auch deshalb wichtig, weil diese die Reihenfolge der Halbkreise in der Ortskurve bestimmen. Deshalb wird immer die schnellere Reaktion mit der höheren Grenzfrequenz in der linken Hälfte und die langsamere in der rechten Hälfte der Ortskurve ausgebildet. [78]

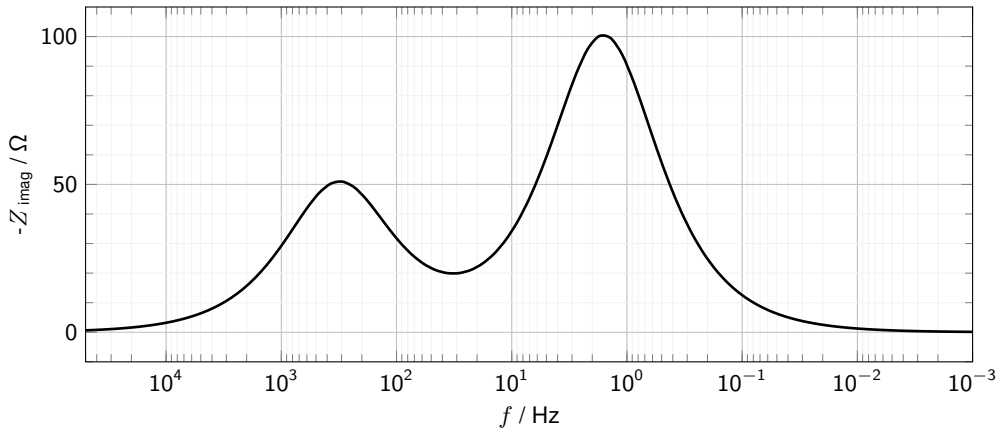

Abbildung 5.25: Imaginärteil über die Frequenz mit den beiden Grenzfrequenzen die sich als Gipfel ausbilden. Zur besseren Zuordnung ist die Abszisse invertiert, womit sich der Frequenzverlauf mit der Ortskurve deckt.

[77] Vgl. Matt Lacey 2022a.
[78] Vgl. Matt Lacey 2022a.

6 MESSAUFBAU UND HARDWARE

Der Messaufbau für die Durchführung der Impedanzspektroskopie wurde in einer separaten Masterarbeit entwickelt [79]. Ziel dieser Entwicklung ist es, die derzeitigen Restriktionen der am Markt verfügbaren Messgeräte zu umgehen. Diese Einschränkungen betreffen nicht nur den Messbereich sondern auch den Wertebereich, den das Messgerät in den Prüfling einprägen kann. Der Eigenbau bietet darüber hinaus noch den Vorteil, dass die Schnittstellen selbst gewählt werden können. Nicht zuletzt kann das System auf den jeweiligen Einsatz angepasst und für zukünftige Aufgaben erweitert werden. Mit dem Aufbau können folgende technische Spezifikationen erreicht werden:

- Frequenzbereich der Einprägung von 1 Hz bis 50 kHz
- Wechselstromamplitude von 20 A bei 50 kHz und 12 A bei 1 Hz
- Gleichstromanteil bis 450 A bei 33 V
- Es wird galvanostatisch (stromgeführt) gemessen

Während der Messung wird der Stack mit einem bestimmten Gleichanteil betrieben. Auf diesen Gleichstrom wird ein wechselstromverstärktes Sinussignal überlagert. Wie Abbildung 6.1 veranschaulicht, erhält man am Knotenpunkt ein zusammengesetztes Signal.

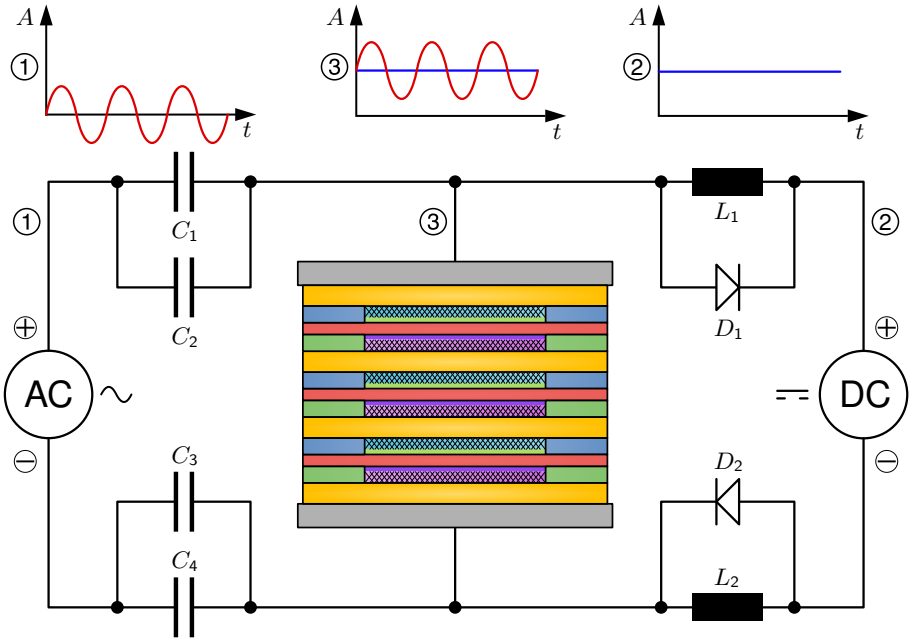

Abbildung 6.1: Koppelschaltung für die Einprägung des zusammengesetzten Signales aus Gleich- und Wechselstromanteil.

Damit dies funktionieren kann, ist eine Koppelschaltung notwendig. Da die Kondensatoren wie eine Leitungsunterbrechung wirken, kann die Gleichstromquelle keinen Strom in die Wechselstromquelle schicken.

[79] Vgl. Joshua Eder 2023.

Die beiden parallelgeschalteten Kondensatoren haben jeweils eine hohe beziehungsweise geringe Kapazität, womit sowohl der hohe wie auch niedrige Frequenzbereich abgedeckt werden kann. Die beiden Spulen auf der Seite der Gleichstromquellen dienen zur Entkoppelung des Wechselstromanteils. Parallel dazu sind zwei Dioden verschaltet, welche im Fall einer plötzlichen Unterbrechung als Freilaufdioden dienen und die in den Spulen gespeicherte Energie abbauen [80]. Durch die Messungen von Spannung und Strom kann über die Analyse des Frequenzspektrums dann auf die Impedanz entlang eines Frequenzbandes geschlossen werden.

Abgesehen von der Elektronik für die Einprägung bedarf es noch weiterer Geräte, um eine Impedanzmessung möglich zu machen. Die höchste Instanz stellt der Teststandsrechner dar. Dieser ist über Ethernet mit der speicherprogrammierbaren Steuerung (SPS) verbunden und darauf läuft die Visualisierungs- und Steuersoftware. Über die SPS werden alle Peripheriegeräte gesteuert und andere Sensordaten wie Drücke und Temperaturen ausgelesen und verarbeitet. Der Applikationsrechner beherbergt die Software e-Gen Spectrolyser, welche die EIS-Messungen koordiniert. Diese wird vom Teststandsrechner initiiert. Die Software ist mit dem X-ion Messgerät verbunden und startet die Datenaufzeichnung. Auf dem Messgerät läuft eine eigene Software zur Erfassung und Verarbeitung der Sensordaten. Über den Applikationsrechner wird auch der Signalgenerator angesteuert, welcher die Einprägefrequenz für die Wechselstromquelle vorgibt. Diese ist ebenfalls über eine USB-Schnittstelle mit dem Applikationsrechner verbunden. Die Gleichstromquelle ist über das CAN-Protokoll direkt mit der SPS verbunden. Um eine synchrone Prozessüberwachung sowie Messdatenaufzeichnung zu gewährleisten, kommuniziert das X-ion Messgerät über eine CAN-Schnittstelle mit der SPS. Die wichtigsten Geräte sowie deren Kommunikationswege und Protokolle sind in Abbildung 6.2 illustriert und für eine genauere Beschreibung wird auf die Masterarbeit [81] verwiesen.

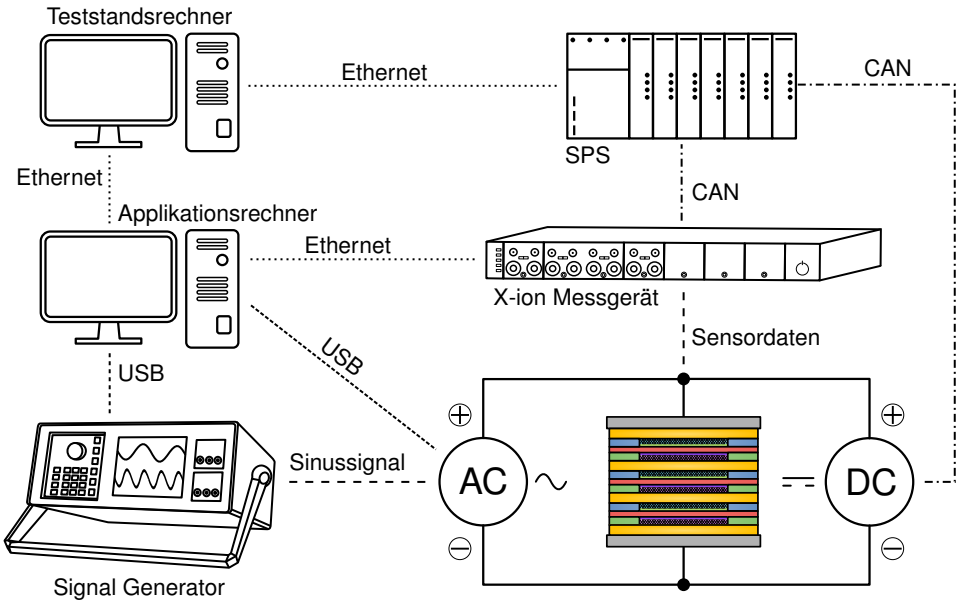

Abbildung 6.2: Schematische Darstellung der beteiligten Instanzen und deren Kommunikation.

[80] Vgl. Joshua Eder 2023, S. 39.
[81] Vgl. Joshua Eder 2023, S. 50–52.

7 KALIBRIERUNG DER SENSORIK

Damit das Impedanzspektrum bestimmt werden kann, muss die Spannung sowie der Strom am Stack gemessen werden. Bei der Vermessung der Zellen ist nur die Zellspannung relevant, da im gesamten Stack ein konstanter Strom fließt und sich nur die Spannungswerte zwischen den Zellen ändern. Somit kann die Strommessung am Stack auch für die Auswertung der Einzelzellen herangezogen werden. Das Kalibrieren der Sensoren sorgt dafür, dass die frequenzabhängigen Eigenschaften der verbauten Komponenten kompensiert werden und diese somit keinen Messfehler erzeugen.

7.1 Kalibrierung des Spannungsteilers

Gemessen wird einmal der Spannungsabfall über den gesamten Stack, sowie der jeweilige Spannungsabfall in den Zellen, wie in Abbildung 7.1 ersichtlich. Da es bei den Zellmessungen nur um den relativen Spannungsabfall geht, ist jede Bipolarplatte mit zwei Voltmetern verbunden.

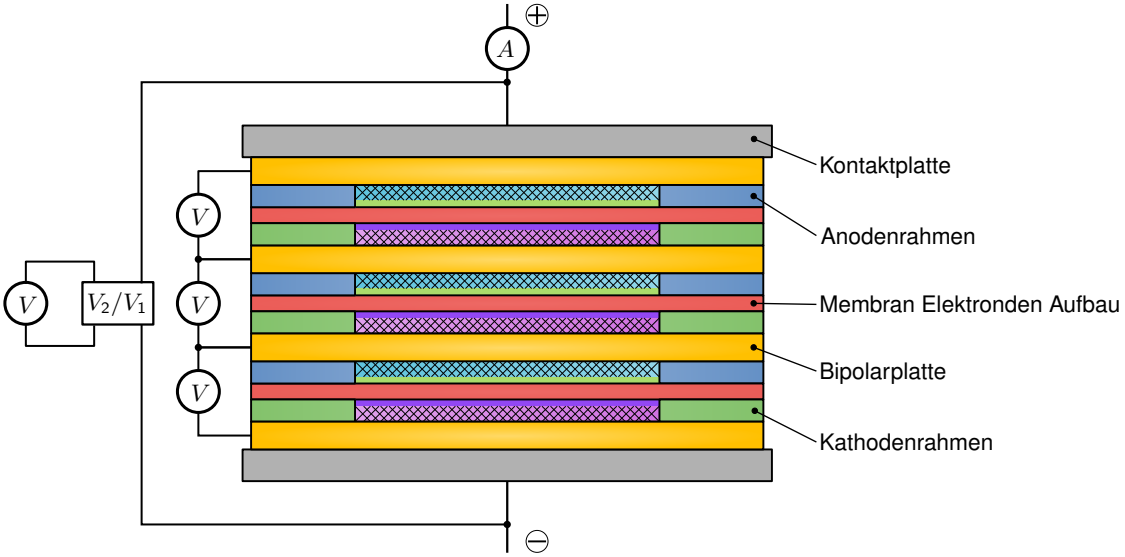

Abbildung 7.1: PEM-Stack mit 3 Zellen und eingezeichneten Messpunkten für die Spannungs- und Strommessung.

Die Messungen werden mit einem X-ion der Firma AVL durchgeführt. Beim Messmodul handelt es sich um das EIS E4X1, wobei die Abtastrate anwenderseitig auf 500 kHz begrenzt wird. Der Eingangsbereich ist jedoch auf ± 10 V limitiert, womit die Spannung am Stack nicht direkt gemessen und nur über die Summe der Zellspannungen ermittelt werden kann. Deshalb ist zwischen der Messstelle und dem Voltmeter der galvanisch getrennte Spannungsteiler Entube DE von VeriVolt zwischengeschaltet, der die Spannung von bis zu ± 1500 V auf ± 5 V oder ± 10 V absenkt. Da der Spannungsabfall immer proportional zum Übersetzungsverhältnis des Spannungsteilers ist, kann auf die Ausgangsgröße zurückgerechnet werden. Dies gilt jedoch nur für einen Frequenzbereich bis circa 20 kHz, wie Abbildung 7.2 verdeutlicht. Wird diese Frequenz überschritten, fällt der Amplitudenwert am Ausgang mit steigender Frequenz immer stärker ab. Unterhalb

dieser Frequenz ist jedoch auch ein nahezu konstanter Fehler zwischen dem Ein- und Ausgangswert ersichtlich. Somit verfälscht der Spannungsteiler das Messergebnis über den gesamten Frequenzbereich, weshalb eine Korrektur notwendig ist. Deshalb wird aus den beiden Messwerten am Ein- und Ausgang der Quotient berechnet, wodurch sich der Skalierungsfaktor bei unterschiedlichen Frequenzen ergibt, wie rechts in Abbildung 7.2 ersichtlich.

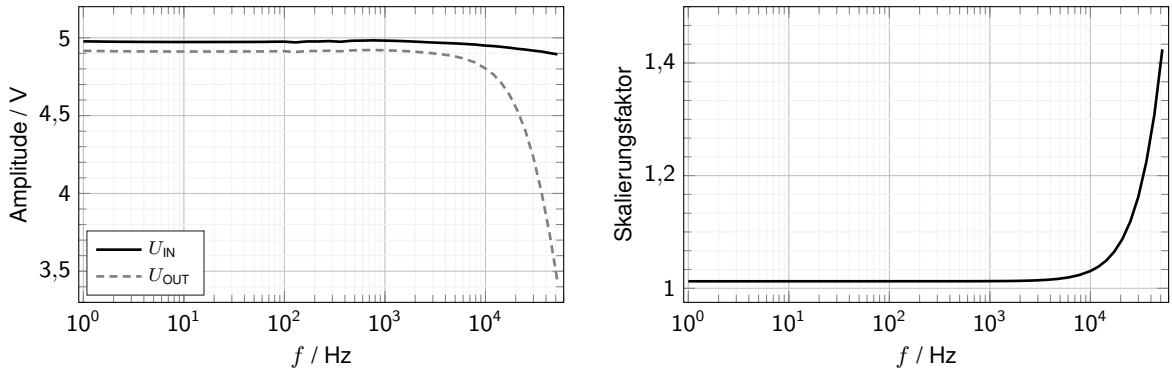

Abbildung 7.2: Spannungsverlauf am Ein- und Ausgang des Spannungsteilers links, sowie der Verlauf des Spannungsverhältnisses über ein Frequenzband.

Der Spannungsteiler beeinflusst nicht nur den Amplitudengang, sondern auch den Phasengang erheblich. Die Messung des Phasenwinkels ist wieder in der rechten Hälfte der Abbildung 7.3 dargestellt.

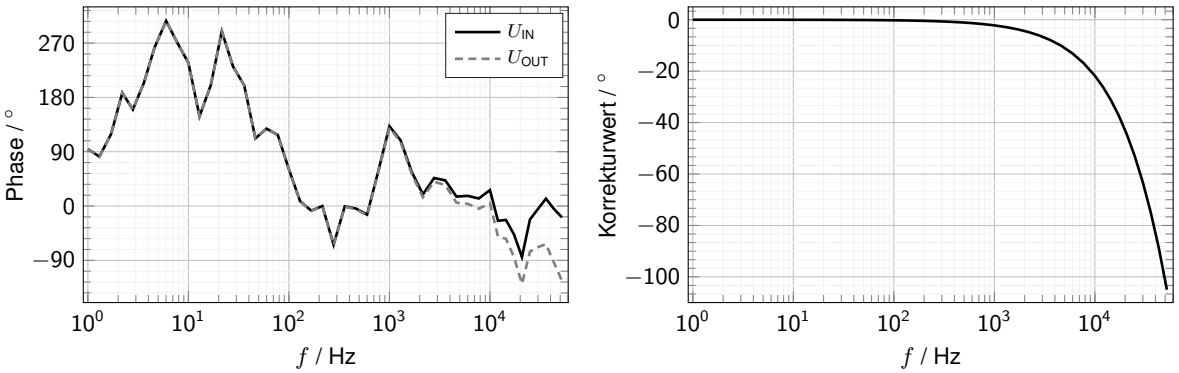

Abbildung 7.3: Phasenverlauf am Ein- und Ausgang des Spannungsteilers links, sowie der Verlauf der Phasendifferenz über ein Frequenzband.

Wie der Korrekturwert in der rechten Bildhälfte zeigt, stellt sich ein Phasengang hier schon bei circa 1 kHz ein. Auch hier gilt wieder, je höher die Frequenz desto größer der Fehler.

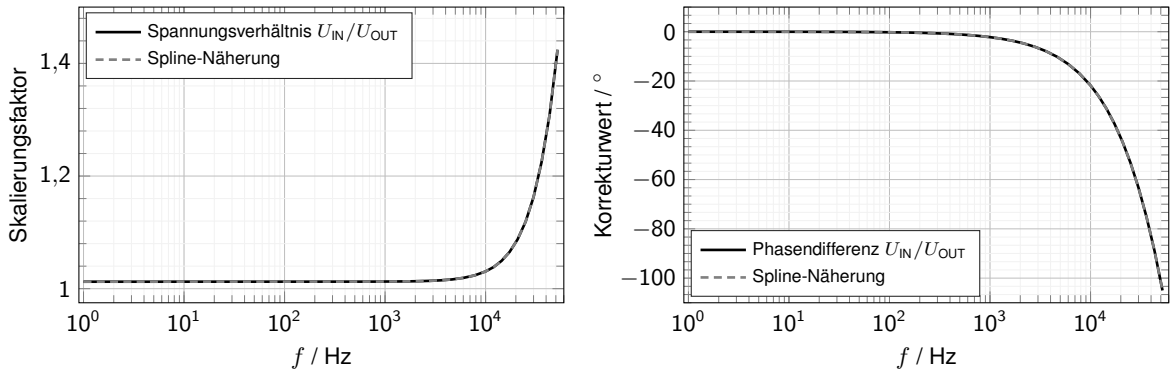

Abbildung 7.4: Spline-Näherung für die Amplitude links sowie für die Phase der Spannung rechts am Spannungsteiler.

Im Gegensatz zur Amplitude ergibt sich aus der Differenz der beiden Messwerte ein Korrekturwert, da der Winkelbereich begrenzt ist und deshalb nicht skaliert werden kann.

Abbildung 7.4 zeigt die Näherung des Skalierungs- und Korrekturwertes mittels Spline. Diese Näherung ist notwendig, da die Faktoren bis zu diesem Zeitpunkt nur für diskrete Frequenzwerte vorliegen. Decken sich diese nicht mit den Frequenzwerten in der eigentlichen Messung, können diese nicht korrigiert werden. Erst durch die Beschreibung anhand einer Spline-Funktion kann der Frequenzbereich kontinuierlich aufgelöst werden, was eine frequenzabhängige Korrektur erst möglich macht.

7.2 Kalibrierung des Stromsensors

Bei der Elektrolyse fließt ein hoher Gleichstrom, weshalb zumeist ein Stromsensor, der nach dem Prinzip des Hall-Effektes arbeitet, zum Einsatz kommt. Dieser misst die magnetische Flussdichte in der Zuleitung, welche sich durch den Stromfluss im Leiter bildet und kann dadurch auf den Strom rückschließen. Somit ist es möglich die Zuleitung durch den Sensor zu führen, wie in Abbildung 7.1 schematisch gezeigt, ohne diese am Sensor anschließen zu müssen. Da dieses Messprinzip eine Frequenzabhängigkeit besitzt und bei der Impedanzspektroskopie ein Wechselstromsignal überlagert wird, bringt der Sensor einen Messfehler ins System ein. Damit dieser Fehler kompensiert werden kann, muss das Verhalten des Sensors über ein definiertes Frequenzband ermittelt werden. Dafür dient ein reines Wechselstromsignal eines Funktionsgenerators, der ein Frequenzband einprägt und somit als Stromquelle dient. Weil der Stromsensor, im Gegensatz zum Funktionsgenerator, für sehr hohe Ströme ausgelegt ist, wird die Zuleitung mehrmals durch den Sensor geführt, wie Abbildung 7.5 illustriert.

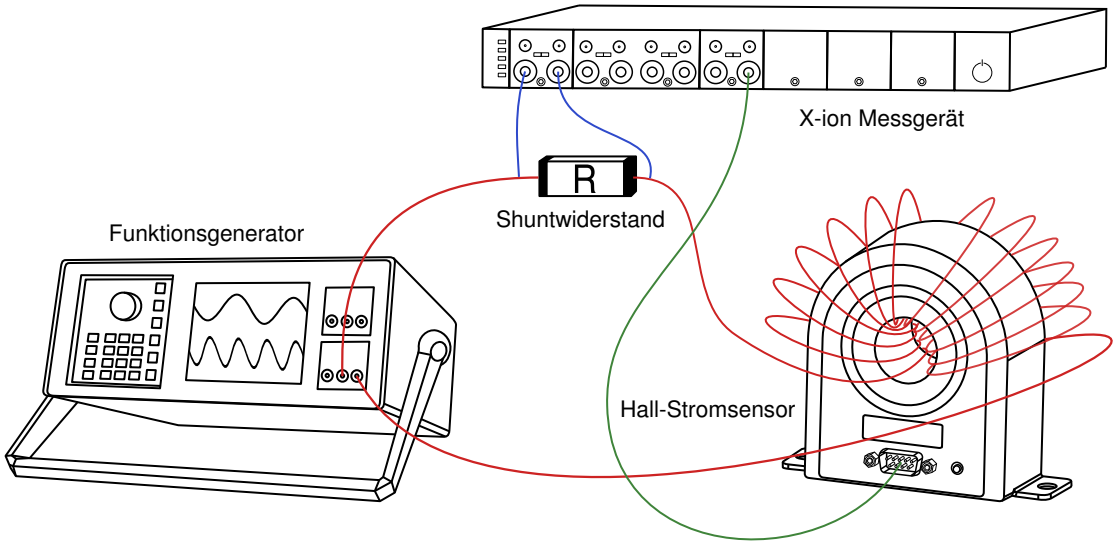

Abbildung 7.5: Messaufbau für die Kalibrierung des Stromsensors mit einem Funktionsgenerator als Signalquelle. Die Zuleitung wird mehrmals durch den Stromsensor geführt und ist in Serie mit dem Messwiderstand verschaltet. Der Spannungsabfall am Widerstand sowie der Sensorausgang werden mit einem X-ion von AVL gemessem.

Der gemessene Strom hängt somit proportional von der Anzahl der Wicklungen ab. Durch den geringen Stromfluss im Leiter kann dieser auch über den Spannungsabfall an einem Messwiderstand bestimmt werden. Dieser hat den Vorteil, dass ein perfekter ohmscher Widerstand kein Frequenzverhalten hat, wie in Kapitel 5.3 schon näher erläutert. Deshalb dient der gemessene Spannungsabfall am Shuntwiderstand als

Referenzwert, mit dem die Stromsensoren verglichen werden. Durch diesen Vergleich kann der Frequenzgang ermittelt und korrigiert werden. Dies wird für zwei Stromwandler durchgeführt. Einmal für einen 300 A Sensor von CAENels und einmal für einen 600 A Sensor von Danisense. Wobei der Sensor von CAENels ein Spannungssignal proportional zum gemessenen Strom ausgibt, das im Verhältnis 1 : 30 steht. Dieses ergibt sich aus dem Messbereich von 0 A - 300 A und dem Bereich des Ausgangssignales von 0 V - 10 V. Im Gegensatz dazu wird vom Sensor von Danisense ein Stromsignal ausgegeben, welches ein Verhältnis von Primär- zu Sekundärseite von 1 : 1500 aufweist. Dieses kann jedoch nicht direkt gemessen werden, da mit dem X-ION Modul EIS E4X1 nur Spannungssignale aufgezeichnet werden können, weshalb das Stromsignal über den Spannungsabfall an einem Messwiderstand ermittelt wird.

Der Frequenzgang der Amplitude ist auf der linken Seite in Abbildung 7.6 dargestellt, es zeigt sich, dass die Sensoren bis zu einer Frequenz von 1 kHz ein nahezu gleiches Verhalten aufweisen.

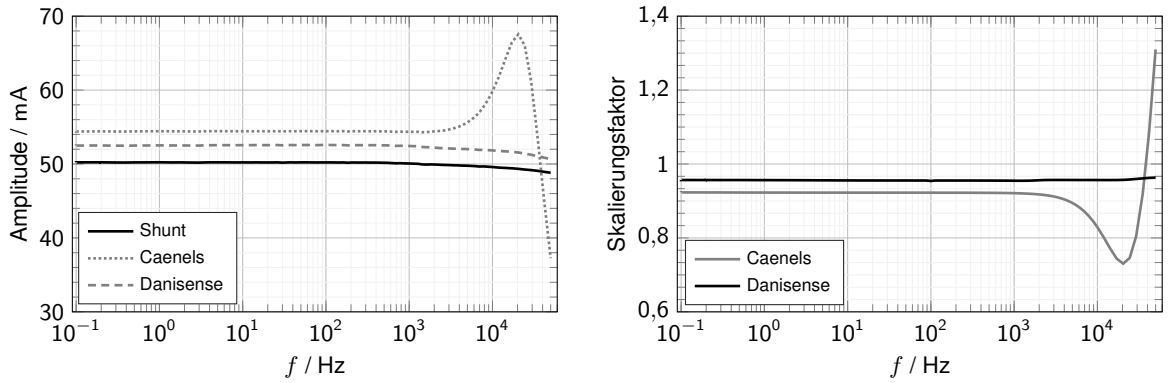

Abbildung 7.6: Amplitudengang für die beiden Stromsensoren sowie dem Shunt-Widerstand als Referenzgröße links. Rechts die beiden Skalierungsfaktoren für die Stromsensoren, die sich aus dem Quotienten zum Shunt ergeben.

Auch der Messfehler bleibt in diesem Bereich zwischen Shunt und den Sensoren weitgehend konstant, wie der Skalierungsfaktor in der rechten Bildhälfte verdeutlicht. Wird die Frequenz weiter erhöht, zeigt sich das der Frequenzgenerator die Stromamplitude nicht mehr halten kann, was sich im degressiven Verlauf der Shunt Messung im hohen Frequenzbereich widerspiegelt. Auch im hohen Verlauf kann der Sensor von Danisense dem Systemverhalten folgen und bringt so keinen zusätzlichen Fehler ins System ein, was wiederum der annähernd gerade Verlauf des Skalierungsfaktors unterstreicht. Im Gegensatz dazu zeigt der Sensor von CAENels ein sehr ausgeprägtes Frequenzverhalten. Hier steigt die gemessene Amplitude zunächst sehr rapide an, um anschließend sehr stark abzufallen. Würde dieses Verhalten nicht kompensiert werden, wäre eine Messung im hohen Frequenzbereich nicht möglich. Durch einen Spline wird der Verlauf des Skalierungsfaktors über die Frequenz angenähert, womit eine kontinuierliche Auflösung für alle gemessenen Frequenzen möglich ist und somit der Messfehler durch den Sensor kompensiert werden kann. Aufgrund des guten Frequenzverhaltens des Danisense wird dieser für die Messungen herangezogen.

Analog zum Frequenzgang der Amplitude ist in Abbildung 7.7 der dazugehörige Phasengang illustriert. Auch hier zeigt sich, dass der Sensor von Danisense ein nahezu identes Messergebnis im Vergleich zum Shunt-Widerstand ausgibt und somit keinen Messfehler in die Messkette einbringt. Das Gegenteil ist wiederum beim Sensor von CAENels der Fall. Der Fehler in der Amplitude bei hohen Frequenzen zeigt sich auch in der Phase wieder, was besonders deutlich in der rechten Bildhälfte beim Korrekturwert ersichtlich ist. Dieser Wert wird bei der Phase jedoch aus der Differenz zur Referenzmessung ermittelt.

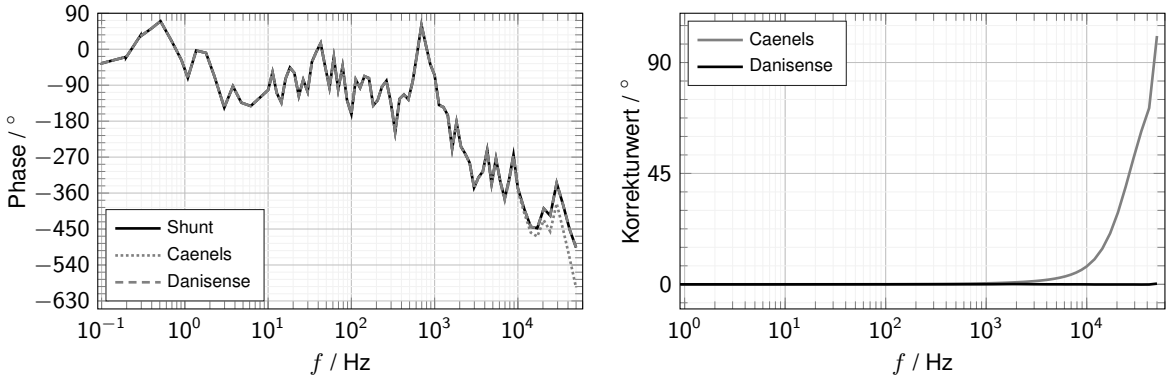

Abbildung 7.7: Phasengang für die beiden Stromsensoren sowie dem Shunt-Widerstand als Referenzgröße links. Rechts die beiden Korrekturwert, die sich aus der Differenz zum Shunt ergeben.

8 ENTWICKLUNG DER AUSWERTUNGSALGORITHMEN

Damit die Messdaten graphisch ausgegeben werden können und damit für den Menschen lesbar beziehungsweise interpretierbar sind, bedarf es einer Abfolge an Programmschritten. Der Datensatz muss eingelesen, verarbeitet, gespeichert und dargestellt werden. Nur wenn alle diese Schritte erfolgreich ablaufen und ineinandergreifen, ist eine Messauswertung möglich.

8.1 Programmiersprache JULIA

Die wohl populärste Programmiersprache im wissenschaftlichen Umfeld und im Ingenieursbereich ist MATLAB. Dieser Stellenwert kommt MATLAB zu, weil es interaktiv genutzt werden kann, viele nummerische Algorithmen bereits implementiert oder als Pakete verfügbar sind und vor allem wegen dem einfachen Darstellen von Daten. JULIA ist eine Hochsprache mit hoher Performance und dynamischer Programmierung. Sie wurde speziell für die Wissenschaft und die technische Datenverarbeitung sowie maschinelles Lernen entwickelt. Ein Just-in-Time (JIT) Compiler der auf der Infrastruktur des LLVM Projektes (Compiler Infrastruktur Projekt) basiert, führt zusammen mit dem Design der Sprache dazu, dass sie nahezu gleich performant wie C und FORTRAN ist. Im Gegensatz zu anderen Sprachen, die eine strikte Abfolge von Schreiben-Compilieren-Ausführen verfolgen, ist JULIA eine interaktive Sprache. So können Codeteile ausgeführt werden und im Speicher verbleiben und fortlaufend Funktionen hinzugefügt werden, was vor allem bei sehr komplexen Algorithmen von Vorteil ist und die Produktivität steigert. Auch Bibliotheken und Funktionspakete können in JULIA sehr einfach eingebunden werden. Von C und FORTRAN können diese sogar ohne zusätzlichen Code-Wrapper inkludiert werden. PYTHON-Code kann mit dem Paket PyCall direkt in der JULIA-Umgebung verwendet werden. Das System hinter den Funktionspaketen ist sehr einfach zu benutzen und auch geschriebene Funktionen können schnell und einfach mit der Gemeinschaft geteilt werden [82].

Funktionspaket	Funktionalität
FFTW.jl	Fourier Transformationen und Signalverarbeitung
WindowFunctions.jl	Filter für Signalverarbeitung
HDF5.jl	Lesen und schreiben von .hdf5 Dateien
MAT.jl	Lesen und schreiben von .mat Dateien
JLD2.jl	Lesen und schreiben von JULIA nativen Dateien
PlotlyJS.jl	Einbinden der Plotfunktionen von plotly.js (Javascript)
ProgressMeter.jl	Anzeigen des Programmfortschrittes in der Ausgabe
StatsBase.jl	Statistik Funktionen
DSP.jl	Funktionen für die digitale Signalverarbeitung
CubicSplines.jl	Interpolation von Daten mit Splines
Glob.jl	Generator für Pfadname
SpecialFunctions.jl	Spezielle mathematische Funktionen

Tabelle 8.1: Auflistung aller im Projekt verwendeten Funktionspakete und deren Funktionalität.

[82] Vgl. Clemens Heitzinger 2022, S. 5–8.

Durch das eingebaute Management System können diese bequem eingebunden, entfernt und aktualisiert werden. Die für dieses Projekt verwendeten Pakete und deren Funktionalität sind in Tabelle 8.1 aufgelistet.

8.2 Programmschritte für Datenauswertung und Programmmodule

Da ein komplettes Programm in einer Datei sehr lang, unübersichtlich und schwer nachvollziehbar ist, bestehen das hier beschriebene aus einer Hauptdatei, die mehrere untergeordnete Dateien einbindet. In diesen Dateien werden alle Funktionen gespeichert, wodurch das Hauptprogramm schlank und übersichtlich gehalten werden kann. Diese untergeordneten Dateien werden in JULIA als Module bezeichnet. Der Unterschied zu den vorher erwähnten Paketen ist, dass Pakete öffentlich verfügbar sind und für die Installation und Wartung eigene sogenannte TOML-Dateien besitzen, wohingegen Module auch lokal gespeichert und eingebunden werden können. Das heißt, Module können als Baublöcke betrachtet werden, da auch Pakete diese beinhalten. Aufgebaut sind diese analog zu Funktionen, wie der Beispielcode 8.1 verdeutlicht.

```
module Name_Modul
using Paket_einbinden
export Beispiel_Funktion
# Definition der Beispiel_Funktion
end
```

Listing 8.1: Beispiel für die Definition und den Aufbau eines Modules.

Das Modul bekommt einen Namen, mit dem dieses aufgerufen werden kann. Anschließend können noch über using Pakete eingebunden werden. Über export können dann Funktionen im Modul zur Verfügung gestellt werden, sofern das Modul im Hauptprogramm eingebunden ist. Auf Funktionen, die dort nicht deklariert sind, kann auch außerhalb des Modules nicht zugegriffen werden. Der Block wird mit einem end abgeschlossen. Analog zu anderen Sprachen werden Funktionen mit function deklariert, wie Beispiel 8.2 zeigt. Über return werden die Rückgabewerte definiert.

```
function Beispiel_Funktion(x,y)
        # Definition des Funktionscodes
        return x,y
end
```

Listing 8.2: Beispiel für die Definition und den Aufbau einer Funktion.

Somit unterscheiden sich diese darin, dass die Module Funktionen als Rückgabewerte liefern und Funktionen Objekte oder Werte zurückgeben. Die drei Module, die für die Datenauswertung und Bereitstellung geschrieben wurden, sind in Abbildung 8.1 horizontal am oberen Bildrand aufgelistet. Außerdem ist dort der gesamte vereinfachte Programmablauf dargestellt. Der erste Schritt besteht darin den Pfad zu deklarieren, in dem die Messdaten abgespeichert wurden. Ein weiterer Pfad gibt den Ort, an dem die Ergebnisse gespeichert werden, an. Im nächsten Schritt wird überprüft, ob die Eingabe der Fensterfunktion korrekt ist und im selben Schritt wird auch die Groß- und Kleinschreibung korrigiert, sofern diese falsch ist. So wird sichergestellt, dass nur korrekte Angaben diesen Schritt passieren und es zu keinen Fehlern in der späteren Signalanalyse, aufgrund von Eingabefehlern, kommt. Anschließend muss die Zeit eingegeben werden, welche vom Beginn der Messdaten weggeschnitten wird, um eine mögliche Einschwingphase des Systems zu kompensieren. Ab diesem Punkt könnte die Signalanalyse schon durchgeführt werden, für die spätere Ausgabe der Grafiken müssen jedoch noch die Plotparameter bestimmt werden. Hierbei werden alle wichtigen Eigenschaften und das Erscheinungsbild der Grafiken definiert. Auch diese Eingaben werden wieder auf ihre Korrektheit überprüft, um einen späteren Programmabbruch aufgrund einer falschen Angabe zu

vermeiden. Die Überprüfung erfolgt jeweils in einer Funktion im Modul Functions for Stack Evaluation, wie auch die gleiche Farbgebung in Abbildung 8.1 verdeutlicht.

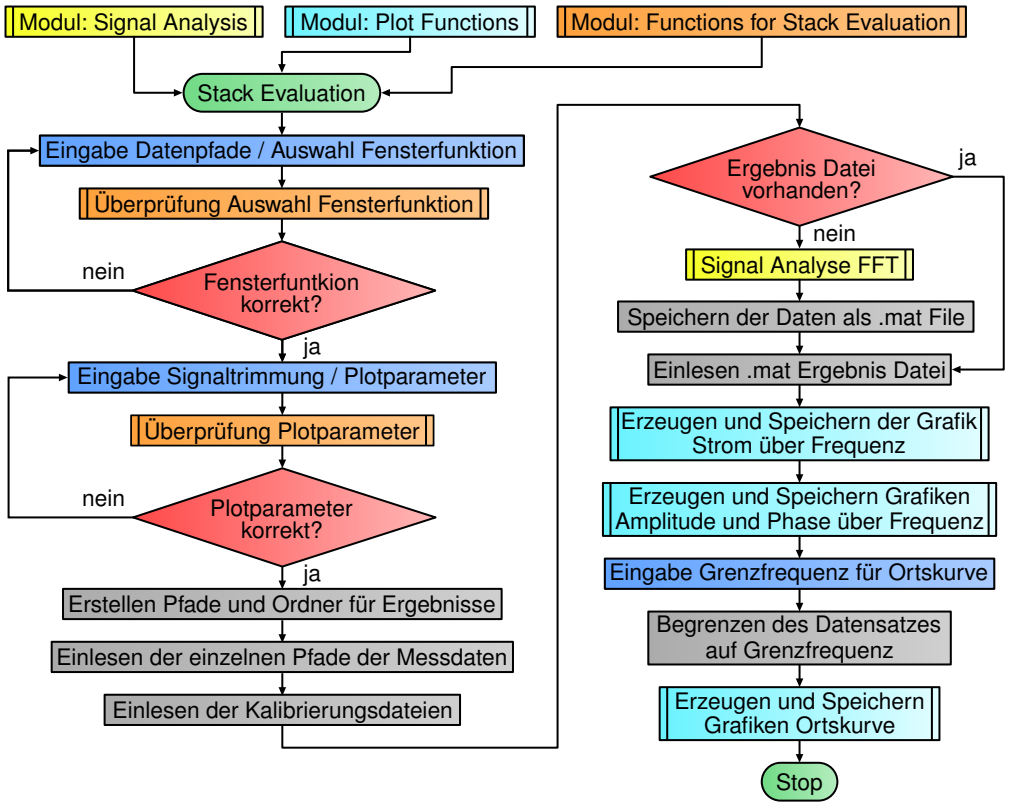

Abbildung 8.1: Programmablauf des Hauptprogrammes Stack Evaluation. In blau die manuellen Eingaben, orange die Unterfunktionen, rot die Entscheidungen.

Im Ergebnispfad wird eine Ordnerhierarchie erstellt, welche die Grafiken von den Ergebnisdateien der Signalanalyse trennt. Zusätzlich werden noch Unterordner für die unterschiedlichen Grafiken erstellt, womit diese nach den dargestellten Ergebnissen sortiert werden. Im Anschluss wird ein Vektor mit allen Pfaden der Messdateien erstellt, wobei sich immer eine Messung des Signalrauschens im Datensatz befindet, die herausgenommen werden muss. Die in Abschnitt 7 berechneten Splines für die Kalibrierung, werden als letzte Eingabe eingelesen und abgespeichert. Da die Frequenzanalyse der Daten sehr zeitaufwändig ist, wird noch überprüft, ob bereits eine Datei mit den Ergebnissen berechnet wurde. Dies ist vor allem deshalb notwendig, weil die Ausgabe der Grafiken mit unterschiedlichen Plotparameter eine häufige Anwendung darstellt und das Skript somit mehrmals durchlaufen wird. Aufgrund der Komplexität der Signalanalyse wird diese in der Erläuterung des gleichnamigen Modules behandelt. Damit auf die Daten auch außerhalb der JULIA-Umgebung zugegriffen werden kann, werden die Ergebnisse im .mat-Format abgespeichert und können somit in MATLAB genutzt werden. Das anschließende erneute Einlesen der Daten ist nur notwendig, wenn zu einem späteren Zeitpunkt die Grafiken erneut ausgegeben werden sollen und die Ergebnisse nicht mehr im Arbeitsspeicher vorliegen. Auf die Erzeugung der Amplituden- und Phasengrafiken wird im Modul Plot Funktionen näher eingegangen. Da für eine ansprechende Darstellung der Ortskurve nicht immer alle Datenpunkte benötigt werden, wird über einen Eingabeparameter die Höchstfrequenz definiert, wie die Programmzeilen 8.3 zeigt. Somit werden nur Datenpunkte abgebildet, die diese Grenzfrequenz nicht überschreiten. Der letzte Schritt in der Programmabfolge ist das Erzeugen und Speichern der Ortskurvengrafiken. Das gesamte Programm ist in Anhang B ersichtlich. Durch die Frequenzanalyse der Messdaten kann die Datenmenge von circa 40 GB auf ungefähr 7 kB reduzieren werden.

```
Border_Freq = 15000
Frequencyvector = Vector{Float64}(undef, 0)
for frequencies in eachindex(FFT_Stack_U_Freq)
    if FFT_Stack_U_Freq[frequencies] <= Border_Freq
        push!(Frequencyvector, FFT_Stack_U_Freq[frequencies])
    end
end
```

Listing 8.3: Ermitteln des Frequenzvektors bis zur Grenzfrequenz.

8.3 Modul Functions for Stack Evaluation

Das Modul Functions for Stack Evaluation stellt alle Funktionen zur Verfügung, die nicht direkt mit der Signalanalyse und der grafischen Darstellung der Ergebnisse zu tun haben. Der hierarchische Aufbau mit den Funktionen ist Abbildung 8.2 dargestellt.

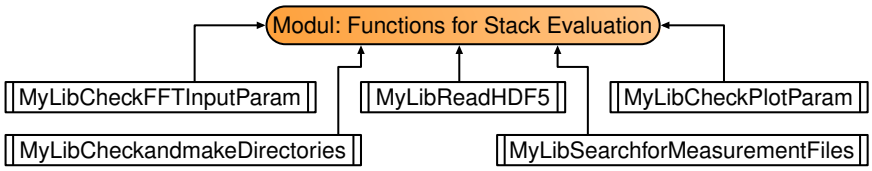

Abbildung 8.2: Hierarchischer Aufbau des Moduls Functions for Stack Evaluation.

Damit eine Iteration durch die einzelnen Messdateien möglich ist, müssen deren Pfade in einem Vektor gespeichert werden, was mit der Funktion MyLibSearchforMeasurementFiles geschieht. Deshalb wird in der ersten Zeile ein Vektor vom Datentyp String initialisiert, mit einer nicht definierten Länge, wie der Programmausschnitt 8.4 zeigt.

```
Raw_Data_Path_Vector = Vector{String}(undef, 0)
Raw_Data_Path = readdir(Raw_Data_Path, join=true)
for loopcount in eachindex(Raw_Data_Path)
        if occursin("_EXC_", Raw_Data_Path[loopcount])
                push!(Raw_Data_Path_Vector, Raw_Data_Path[loopcount])
        end
end
```

Listing 8.4: Ausschnitt aus der Funktion für die Erstellung des Pfadvektors der Messdaten.

Mittels Schleife wird über alle Dateien beziehungsweise alle Dateinamen im Ordner iteriert. Die Benennung der Messdateien folgt dem Aufbau in Abbildung 8.3.

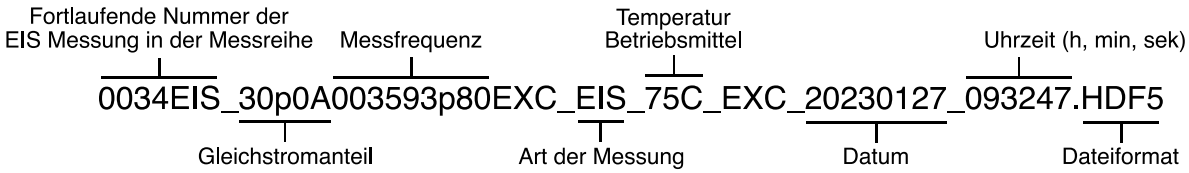

Abbildung 8.3: Aufbau der fortlaufenden Benennung der Messdateien.

Mit der Funktion occursin wird überprüft, ob der String "_EXC_" in der Benennung der Datei vorkommt, da die unerwünschte Messdatei mit "_NEX_" benannt ist. Mittels push! werden alle Pfade der Messdateien im

Vektor abgelegt. Die Überprüfung der Eingabeparameter für die Grafiken erfolgt in der Funktion MyLibCheckPlotParam, sowie die Eingabe der Parameter für die FFT-Analyse werden mit MyLibCheckFFTInputParam untersucht. In beiden Fällen erfolgt dies analog zum Code-Ausschnitt 8.4 mit einem Stringvergleich mittels occursin. Da das Erstellen der Ordnerhierarchie mit MyLibCheckandmakeDirectories trivial ist, wird darauf nicht näher eingegangen, kann aber im Quellcode im Anhang C nachvollzogen werden. Das Einlesen der Messdaten im HDF5-Format erfolgt über die Funktion MyLibReadHDF5. Dabei steht das HDF für Hierarchical Data Format, welches für die Speicherung großer Datenmengen konzipiert wurde. Der Vorteil liegt darin, dass Tabellen und andere Daten in ein und derselben Datei gespeichert werden können und die Verzeichnisstruktur beliebig skalierbar ist. Ein Teil dieser Struktur ist in Abbildung 8.4 zu sehen, wobei es sich im Ordner info_Channel_Data um Messdaten und im Ordner info_DAQ um Informationen über Sensorik und Elektronik handelt. Somit ist es möglich auch den Messaufbau und die Gegebenheiten während der Messung zu archivieren und diese damit nachvollziehbar zu machen.

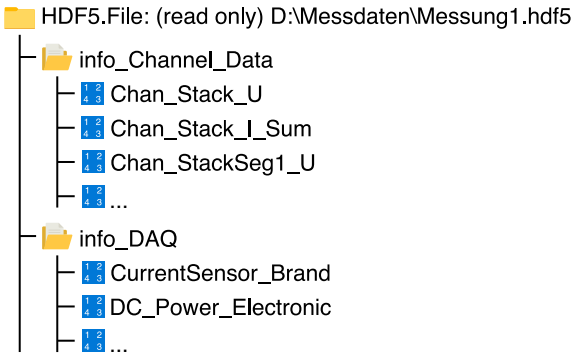

Abbildung 8.4: Auszug aus der Dateistruktur einer Messdatei im HDF5-Format.

Ausgelesen werden die Daten mit Hilfe des Paketes HDF5.jl, wodurch die Dateien über ihren Pfad und die Dateistruktur angesprochen werden können, wie der Programmauszug 8.5 exemplarisch zeigt.

```
h5open(Filepaths, "r") do fid
        Stack_U = read(fid, "info_Channel_Data/Chan_Stack_U")
end
```

Listing 8.5: Einlesen der Daten mittels Dateipfad und Dateistruktur.

Da das Programm universell für unterschiedliche Größen von Stacks mit abweichender Zellanzahl funktionieren soll, muss die Anzahl der Zellen aus der Datei ermittelt werden. Hierfür wird der Ordner info_Channel_Data eingelesen und dessen Inhalt in einen Stringvektor konvertiert. Anschließend wird mittels Schleife über diesen Stringvektor iteriert, wobei bei jedem Durchlauf der Index des Vergleichsstrings erhöht wird. Über einen Vergleich wird gezählt wie viele Zellen in den Messdaten vorhanden sind, wie der Code 8.6 deutlich macht.

```
_data = h5read(Filepaths, "info_Channel_Data")
_data_names = convert.(String, keys(_data))
_segment_counter = 0
for index in eachindex(_data_names)
        if sum(occursin.(String("Chan_StackSeg" * string(index) * "_U"), _data_names)) > 0
                _segment_counter = _segment_counter + 1
        end
end
```

Listing 8.6: Ermitteln der Zellanzahl aus den Messdaten.

Mit der Anzahl der Zellen können deren Messdaten ausgelesen und in einer Matrix gespeichert werden. Hier muss jedoch abgefragt werden, ob es sich um die erste Iteration in der Schleife handelt, da die Funktion hcat erst funktioniert, sobald die Matrix mindestens einen Eintrag besitzt, wie auch die Programmzeilen 8.7 unterstreichen.

```
StackSeg_U = Matrix{Float64}(undef, 0, 0)
for loopcounter in 1:_segment_counter
        if loopcounter == 1
                StackSeg_U = vec(_data["Chan_StackSeg1_U"])
        else
                _segment_name       = String("Chan_StackSeg" * string(loopcounter) * "_U")
                StackSeg_U          = hcat(StackSeg_U, _data[_segment_name])
        end
end
```

Listing 8.7: Einlesen der Zelldaten und speichern in eine Matrix.

Diese Funktion bildet den Abschluss der Datenvorbereitung bevor mit der Signalanalyse fortgefahren werden kann.

8.4 Modul Signal Analysis

Nachdem alle Daten und Parameter für die Signalanalyse bereitgestellt wurden, wird eine Frequenzanalyse durchgeführt. Dabei sind nur die Funktionen MyLibTrimInputSignal, _MyLibWindowFunc und MyLibFFT im Modul gespeichert, alles andere wird in der Hauptfunktion berechnet, wie Abbildung 8.5 zeigt.

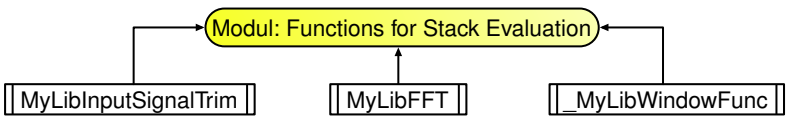

Abbildung 8.5: Hierarchischer Aufbau des Moduls Signal Analysis.

In der Hauptfunktion wird durch den vorher erstellten Vektor mit den Dateinamen der Messdateien iteriert, wobei die restliche Berechnungszeit und der Fortschritt mit dem Funktionspaket ProgressMeter im Terminal ausgegeben beziehungsweise grafisch dargestellt wird. Bei jeder Iteration wird ein Messfile aufgerufen und die Parameter Abtastfrequenz, Zellspannungen, Stackspannung, Summenstrom am Stack, Gleichstrom am Stack, aktive Zellfläche und wie viele Zellen pro Messsegment vermessen werden, ausgelesen. Durch Bilden des Mittelwertes und dessen Subtraktion mit jedem Messpunkt, wir der Gleichspannungsanteil entfernt. Anschließend erfolgt eine initiale Frequenzauswertung der gesamten Messdauer, um die Signalfrequenz zu bestimmen. Da sich hier unvollständige Perioden in der Messung befinden können, wird ein falscher Amplitudenwert ausgegeben. Für die Ermittlung der Frequenz ist jedoch nur dessen Position im Vektor von Belang, wie im Abschnitt 4.3.5 näher beschrieben. Die genaue Frequenz des Signales ist deshalb so wichtig, weil anhand dieser die Beschneidung des Messsignales erfolgt, wie im Unterkapitel 8.4.1 ausgeführt. Nachdem das Signal auf eine volle Periodenanzahl beschränkt wurde, wird eine erneute Frequenzanalyse durchgeführt. Aus dieser können dann die Signalparameter bestimmt und gespeichert werden, wie der Codeausschnitt 8.8 der beschriebenen Schritte zeigt.

```
# Read Data from E-File
global _Sampling_Freq, _Cells_U, _Stack_U, _Stack_I_Sum, _Stack_I_DC, Active_Cell_Area,
Cells_per_Segment = MyLibReadHDF5(Raw_Data_Path_Files[i_file])
# Centering Raw Data around mean for first Analysis
```

```
_Stack_U_Estimation = vec(_Stack_U .- mean(_Stack_U))
# Determine Frequency of Raw Data
_freq_Stack_U, _, _amp_Stack_U, _, _, _ =
MyLibFFT(_Stack_U_Estimation, _Sampling_Freq, 0, FFT_window_func, 0)
# Find Peakposition
_index_ampl_Stack_U = argmax(vec(_amp_Stack_U))
# Trim Signal to whole Periods
_Trimmed_Stack_U = MyLibInputSignalTrim(_Sampling_Freq, _freq_Stack_U[_index_ampl_Stack_U],
                                        _Stack_U, FFT_Trim_of_Signal)
# FFT for Signal Analysis with whole Periods
_freq_Stack_U, _, _amp_Stack_U, _angle_Stack_U, _, _offset_Stack_U =
MyLibFFT(_Trimmed_Stack_U, _Sampling_Freq, 0, FFT_window_func, 0)
# Find Peakposition
_index_ampl_Stack_U = argmax(vec(_amp_Stack_U))
# Safe Results in Vector
push!(FFT_Stack_U_Ampl, _amp_Stack_U[_index_ampl_Stack_U])
push!(FFT_Stack_U_Angle, _angle_Stack_U[_index_ampl_Stack_U])
push!(FFT_Stack_U_Freq, _freq_Stack_U[_index_ampl_Stack_U])
```

Listing 8.8: Abfolge der Programmschritte für die Signalanalyse. Der Code wurde auf die Auswertung der Stackspannung begrenzt. Somit fehlen der Summenstrom sowie der Gleichstrom am Stack in dieser Auflistung, wobei für diese dieselben Programmschritte nötig sind.

Analog dazu erfolgt die Auswertung der Zellen, wobei dieser Vorgang für jede Zelle durchlaufen werden muss.

Nachdem alle Messdaten ausgewertet wurden, müssen diese, wie in Kapitel 7 aufgezeigt, um den Fehler, der durch den Messaufbau ins System eingebracht wird, ausgeglichen werden. Damit man Stacks unterschiedlicher Baugröße oder Form miteinander vergleichen kann, müssen die Messergebnisse noch mit der Zellfläche normiert werden, wie die Programmzeilen 8.9 zeigt.

```
Amplitude_Resistance_Stack = FFT_Stack_U_Ampl_calibrated ./ FFT_Stack_I_Sum_Ampl .*
(Active_Cell_Area / Cells_per_Segment)
```

Listing 8.9: Korrektur des Messfehlers und Normierung auf Zellfläche.

Entgegen der Intuition muss die Fläche mit dem Widerstand multipliziert werden, womit sich die Einheit $\Omega\,\mathrm{cm}^2$ ergibt. Dies ist deshalb der Fall, weil nicht der Strom durch den Stack sondern die Stromdichte bezogen auf die Fläche in $\mathrm{A\,cm}^{-2}$ angegeben wird. Dies lässt sich mit Hilfe des ohmschen Gesetzes nachvollziehen, indem man die Einheiten mit der Notation nach DIN 1313 [83] einsetzt

$$[R] = \frac{[U]}{[I]} = \frac{\mathrm{V}}{\mathrm{A}} = \Omega \quad \text{mit Stromdicht} \quad J = \mathrm{A\,cm}^{-2} \quad \Longrightarrow \quad [R] = \frac{\mathrm{V}}{\mathrm{A\,cm}^{-2}} = \frac{\mathrm{V}}{\mathrm{A}}\,\mathrm{cm}^2 = \Omega\,\mathrm{cm}^2. \tag{8.1}$$

Man kann sich dies aber auch über die Definition des Widerstandes mit

$$R = \rho\,\frac{l}{A} \tag{8.2}$$

herleiten. Angenommen der spezifische Widerstand des Materials ρ und die Länge des Leiters, was in diesem Fall die Dicke des Membran-Elektroden-Aufbaues entspricht und aufgrund der geringen Schichtdicken nur einen vernachlässigbaren Einfluss auf den Widerstand hat, sind konstant. So zeigt sich der proportionale Zusammenhang zwischen Widerstand und Fläche. Da sich der Widerstand bei doppelter Fläche halbiert, muss dieser mit der Fläche multipliziert werden, um ein normiertes und damit vergleichbares Ergebnis zu erhalten. Eine weitere Korrektur ist nötig, sofern mehrere Zellen des Stacks mit einem

[83] Vgl. Rohde & Schwarz 2017, S. 11.

Messsegment erfasst wurden. Im letzten Schritt der Signalanalyse werden die Ergebnisse in Vektoren für die grafische Darstellung umgerechnet. Durch das Speichern im .mat Format können diese so auch außerhalb der JULIA-Umgebung eingelesen und dargestellt werden. Abbildung 8.6 zeigt nochmals grafisch den Programmablauf der Signalanalyse. Der gesamte Code des Modules ist in Anhang D ersichtlich.

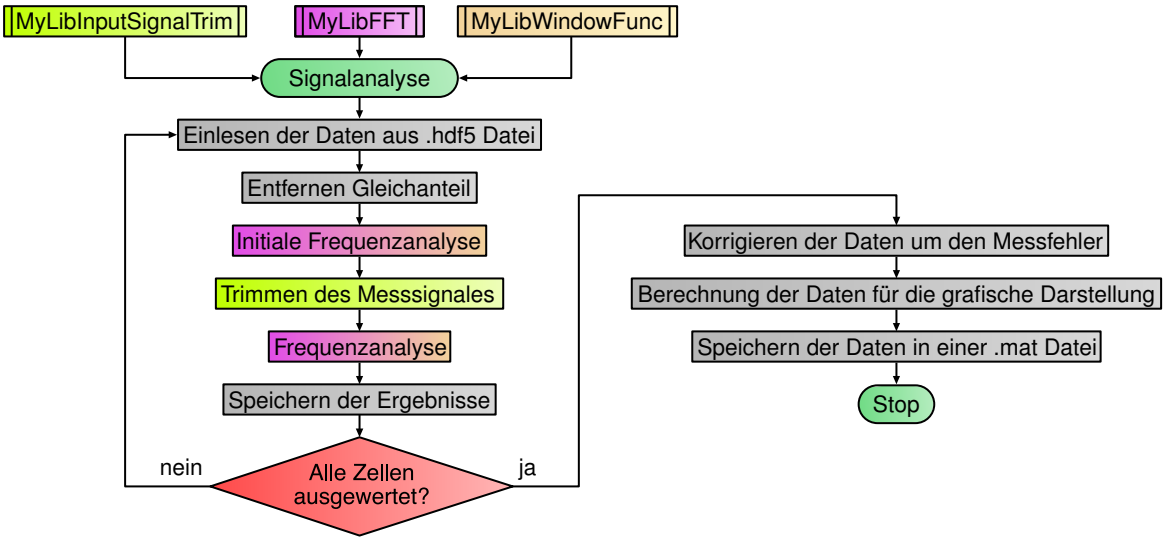

Abbildung 8.6: Ablaufplan der Signalanalyse.

8.4.1 Trimmen des Messsignales

Wie im vorigen Absatz kurz angedeutet, wird das Messsignal beschnitten, um die Einschwingphase des Systems von der Signalanalyse auszuschließen. Deshalb wird über eine manuelle Eingabe ein Wert in Sekunden definiert, welcher dann vom Beginn des Messsignales weggeschnitten wird. Da die Datenpunkte über die Abtastrate mit der Zeitdomäne gekoppelt sind, wird diese mit dem Sekundenwert multipliziert, wodurch man die Anzahl der zu löschenden Punkte in den Messdaten erhält, wie die Codezeile 8.10 verdeutlicht.

```
# Cut away specific Time from the beginning of the Signal
_Signal = _Signal[trunc(Int, _Sampling_Freq * _Trim_of_Signal) : end]
```

Listing 8.10: Wegschneiden der Einschwingphase mittels Parameter in Sekunden.

Damit das Signal eine volle Anzahl an Perioden aufweist, muss über die Abtastrate und die Signalfrequenz die Anzahl der Punkte pro Periode ermittelt werden. Ist diese bekannt, lässt sich über die verbleibende Anzahl an Datenpunkten in der Messung die Anzahl an ganzen Perioden ermitteln, mit der Codeausschnitt 8.11 illustriert.

```
_Number_of_Points_per_Period = Int(_Sampling_Freq / _Frequency)
_Number_of_Periods = length(_Signal) / _Number_of_Points_per_Period
_Trimmed_Signal = _Signal[end - (_Number_of_Periods * _Number_of_Points_per_Period - 1) : end]
```

Listing 8.11: Berechnung der Anzahl an Punkten pro Periode.

Diese Anzahl an Datenpunkten wird vom Ende hergezählt, womit eine Differenz zur vollen Periode ebenfalls weggeschnitten wird, wie Abbildung 8.7 illustriert. Zusätzlich wird überprüft, ob die verbleibenden Messpunkte mindesten vier Perioden beinhalten. Anschließend werden die beschnittenen Messdaten mit ausschließlich voller Periodenanzahl zurückgegeben und können der Frequenzanalyse übergeben werden.

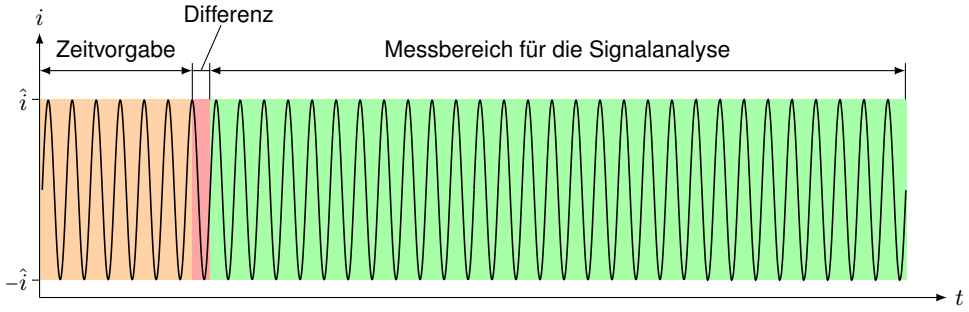

Abbildung 8.7: Trimmen der Messdaten auf ganze Perioden und Wegschneiden der Einschwingphase.

8.4.2 Frequenzanalyse des Messsignales

Die Analyse des Messsignales findet in der Funktion MyLibFFT statt. Hierbei dienen die beschnittenen Messdaten, die Abtastfrequenz, Fensterlänge, Fenstertyp und die Fensterüberlappung als Eingabeparameter. Die Fensterlänge beschreibt dabei die Anzahl der Messpunkte, die sich im Fenster befinden. Die Höchstanzahl an Punkten muss dabei der Anzahl an Messpunkten entsprechen. Nach untenhin ist eine theoretische Grenze von einem Punkt möglich, dadurch kann das Signal jedoch nicht mehr ausgewertet werden, deshalb ist eine sinnvolle Periodenanzahl zu wählen. Wie in Abbildung 8.8 ersichtlich, können die Fenster auch überlappen und so ineinander übergehen.

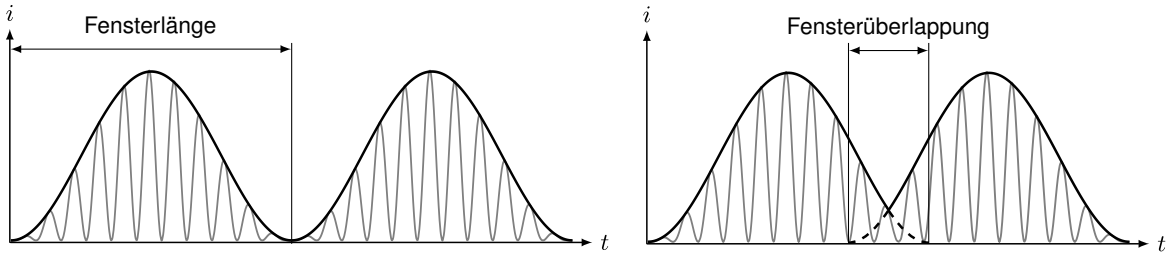

Abbildung 8.8: Fensterlänge und Fensterüberlappung dargestellt mit der Fensterfunktion Hann.

Nachdem das Messsignal mit der entsprechenden Fensterfunktion in der Hilfsfunktion _MyLibWindowFunc skaliert wurde, kann mittels Abtastrate und Anzahl der Messpunkte die Signaldauer berechnet werden. Mit dieser und der Anzahl der Datenpunkte wird dann der Frequenzvektor errechnet, wie der Codeausschnitt 8.12 zeigt.

```
win_func = _MyLibWindowFunc(_window_type, trunc(Int, _window_length))
t_win_duration = _window_length / _sampl_rate
_frequencies = 1 / t_win_duration * (0 : (ceil(Int, _window_length / 2) - 1))
```

Listing 8.12: Berechnung des Frequenzvektors für die Signalanalyse.

Wie in Abschnitt 4.3.5 erwähnt, muss das Spektrum um dessen Mitte gespiegelt werden, weshalb der Mittelpunkt des Datensatzes berechnet wird. Anschließend erfolgt die FFT, wobei hier die Funktion rfft zum Einsatz kommt. Diese ist für reelle Arrays optimiert und beansprucht deshalb nur den halben Speicherplatz, bei gleichzeitig doppelter Geschwindigkeit. Dies wird erreicht, da die Transformation eine konjugierte Symmetrie besitzt und diese genutzt werden kann [84].

[84] Vgl. FFTs Paket Dokumentation 2023.

```
range_ = range(1, step=1, stop=ceil(Int, _window_length / 2))
_FFTcmplx = rfft(win_func .* _data_vec[move_ind])
_FFTcmplx = _FFTcmplx[range_]# use only half Spectrum for Analysis
_amplitudes = (2 / sum(win_func)) .* abs.(_FFTcmplx)# calc amplitude of complex pointer
_angles = vec(angle.(_FFTcmplx));    # calc angle of complex pointer
offset = abs.(_FFTcmplx[1, ])/sum(win_func)
```

Listing 8.13: Berechnung des Frequenzvektors für die Signalanalyse.

Das Ergebnis der Fourier-Transformation wird gespiegelt, indem nur die Hälfte der Datenpunkte weiterverarbeitet wird. Der Amplitudenwert ergibt sich dann aus dem Absolutwert des komplexen Ergebnisses. Aufgrund der Spiegelung müssen alle Werte verdoppelt und mit der Anzahl der Datenpunkte normiert werden, um ein korrektes Ergebnis zu erhalten. Die Winkel werden mit der angle Funktion ermittelt. Im letzten Schritt wird der Offset noch aus dem ersten komplexen Ergebnis aus dessen Absolutwert berechnet und wieder mit der Anzahl an Datenpunkten normiert, wie der Codeausschnitt 8.13 zeigt.

8.5 Modul Plot Functions

Nachdem die Signalanalyse abgeschlossen und die Ergebnisse gespeichert wurden, gilt es diese darzustellen. Aufgrund der Vielzahl an möglichen Darstellungsvarianten, fällt die Hierarchie des Modules Plot Functions dementsprechend umfangreich aus, wie Abbildung 8.9 verdeutlicht.

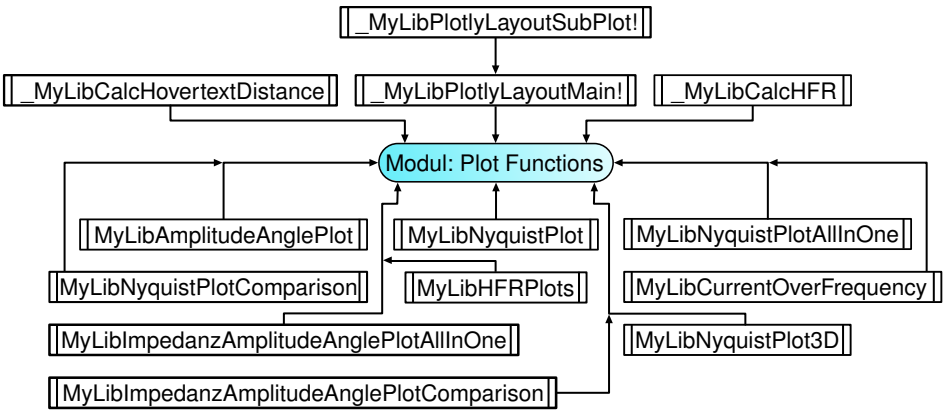

Abbildung 8.9: Hierarchischer Aufbau des Moduls Plot Functions.

Wie in Abschnitt 8.2 schon erwähnt, sollen die Grafiken in unterschiedlichen Dateiformaten und für verschiedene Anwendungen optimiert und automatisiert ausgegeben werden. Deshalb werden im Hauptprogramm die erforderlichen Plotparameter abgefragt, wie der Programmausschnitt 8.14 zeigt.

```
Dataformat                      = ".svg" # plot format, one of: ".html", ".pdf", ".svg"
Transparent_Background          = true
Transparent_Background_Legend   = false
Plot_Size                       = true   # true for whole page and false for half page
Show_Grid                       = true
Show_Meta_Information           = true   # show frequency information in nyquist plot
Calc_HFR                        = true  # calculation of high frequency resistance
```

Listing 8.14: Abfrage der erforderlichen Plotparameter.

Neben dem Dateiformat kann gewählt werden, ob der Hintergrund der Grafik sowie der dazugehörigen Legende transparent ausgeführt wird. Bei der Größe kann zwischen der Breite einer halben und ganzen

DIN A4 Seite gewählt werden. Außerdem ist es möglich, das Gitternetz im Abbildungshintergrund auszublenden. Bei der Metainformation handelt es sich um die Frequenzinformation in der Ortskurve, die im Unterkapitel 8.5.2 näher erläutert wird. In der letzten Auswahlmöglichkeit kann angeben werden, ob es um ein volles oder kurzes Spektrum handelt, was für die Berechnung des Hochfrequenzwiderstandes entscheidend ist. Alle Parameter gelten für alle Formate mit Ausnahme von .html Dateien, da diese interaktive Elemente beinhalten und nur zur Analyse und Bewertung dienen.

Die Darstellung des Amplituden- und Phasenplots gespeichert im .svg Format mit transparentem Hintergrund, Gitterlinien und optimiert für eine halbe bzw. ganze Seite ist in Abbildung 8.10 ersichtlich.

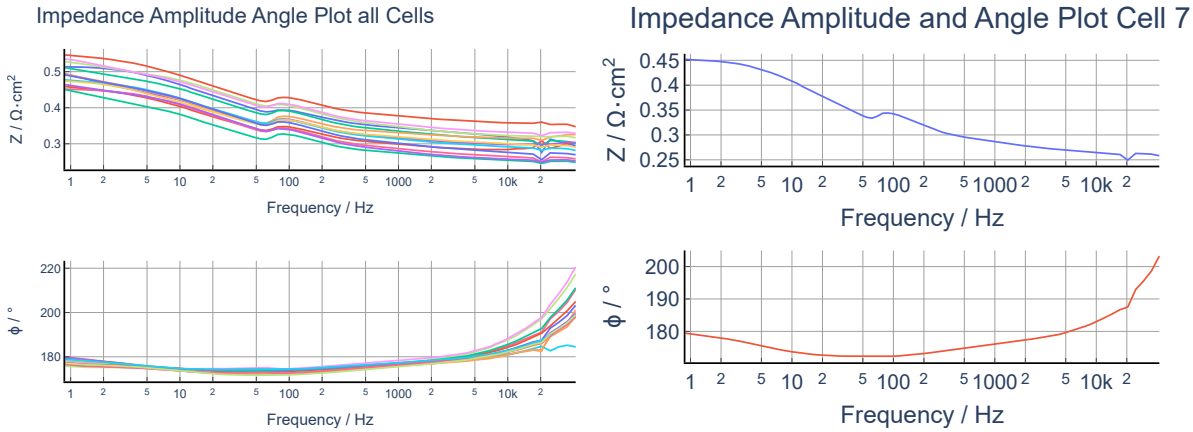

Abbildung 8.10: Amplituden- und Phasenplot optimiert für eine ganze Seite links bzw. für eine halbe Seite rechts. Durch das Anpassen der Schriftgrößen kann ist die Grafik auch in halber Größe gut lesbar.

Gespeichert wird neben allen Einzelzellen auch noch eine Grafik mit allen Zellen in einer Abbildung, wie oberhalb ersichtlich, sowie eine Grafik mit dem Amplituden- und Phasengang des gesamten Stacks. Analog dazu wird auch noch der Strom durch den Stack über die Frequenz dargestellt.

Da die Frequenzinformation in Abbildung 8.10 ersichtlich ist, wird diese nicht zusätzlich eingeblendet. Anders ist dies bei der Darstellung der Ortskurve. Hier ist diese essenzielle Information nicht auf einer Achse aufgetragen. Deshalb werden die dekadischen Frequenzen, sofern der Parameter Show_Meta_Information auf true steht, zusätzlich angegeben, wie Abbildung 8.11 zeigt.

8.5.1 Berechnung der dekadischen Frequenzinformation

Die Berechnung der dekadischen Frequenzen erfolgt in der Plotfunktion für die Ortskurve. Hierfür wird ein Vektor mit den dekadischen Grenzen erstellt. Anhand dieses Vektors wird dann durch die Frequenz der Messung iteriert und wenn eine Frequenz die Grenzfrequenz übersteigt gespeichert. So ergibt sich ein Vektor mit den dekadischen Frequenzen der Messpunkte, wie der Programmausschnitt 8.15 zeigt.

```
_Frequency_Positions = Vector{Int}(undef, 0)
_Frequency_Decade = 10 .^ (range(0 ,step = 1.0, stop = 5)) # Border Frequencies for every Decade
for loopcounter in eachindex(_Frequency_Decade)
        if isa(findfirst(isone, _Frequencies .> _Frequency_Decade[loopcounter]), Number) == true
                push!(_Frequency_Positions,
                findfirst(isone, _Frequencies .> _Frequency_Decade[loopcounter]))
        end
end
```

Listing 8.15: Definieren des dekadischen Frequenzvektors.

Anschließend muss noch die Position der dekadischen Messpunkte gespeichert werden. Die Position, an der die Information in der Grafik abgebildet wird, muss zusätzlich definiert werden. Da ein fixer Abstand zu den Messpunkten in beide Koordinatenrichtungen bei halbkreisförmigen Datensätzen keine zufriedenstellende Lösung darstellt, wird jede Position einzeln errechnet. Dies erfolgt in der Unterfunktion _MyLibCalcHovertextDistance, wobei die Punkte vor und nach dem Punkt mit der dekadischen Frequenz betrachtet werden. Durch diese wird eine Gerade gezogen, deren Mittelpunkt berechnet und anschließend ein Punkt orthogonal zu dieser Linie ermittelt, wie Abbildung 8.11 verdeutlicht. Der Abstand zwischen diesem Punkt und der Linie ist abhängig von der Skalierung der Achsen. So kann gewährleistet werden, dass der Abstand zu den Punkten unabhängig von den dargestellten Messgrößen immer im selben Verhältnis steht.

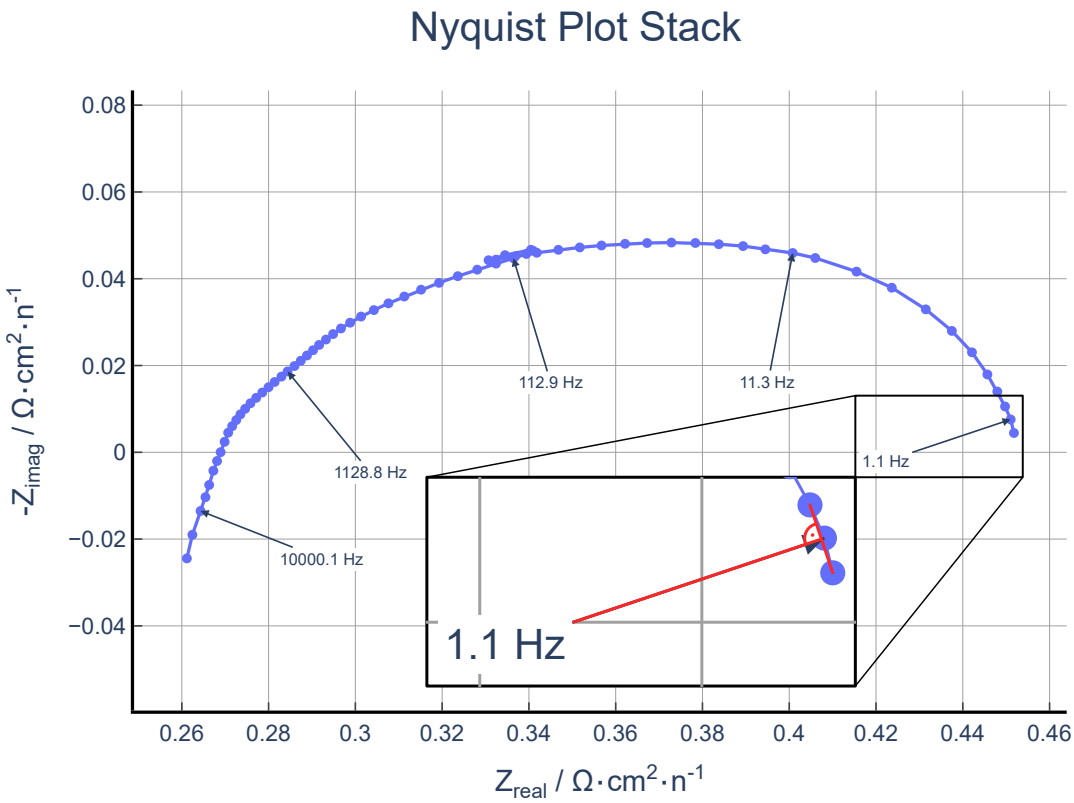

Abbildung 8.11: Ortskurve eines Stacks mit dekadischer Frequenzinformation und ohne Berechnung des Hochfrequenzwiderstandes. Die Daten sind normiert auch die Zellfläche sowie die Zellanzahl und können so mit Messung von Einzelzellen verglichen werden. Zusätzlich ist die Berechnung der Position der Zusatzinformation vergrößert grafisch dargestellt.

8.5.2 Berechnung des Hochfrequenzwiderstandes und dessen Frequenz

Da der Hochfrequenzwiderstand am Nulldurchgang der Ordinate abgelesen wird, muss dieser Schnittpunkt mit dem jeweils ersten positiven sowie negativen Punkt ermittelt werden. Deshalb wird zwischen diesen beiden Punkten eine Linie gezogen, anhand der man den Schnittpunkt exakt ermitteln kann, wie in Abbildung 8.12 illustriert. Da auch die Frequenz an diesem Punkt von Interesse ist, wird auch diese berechnet. Dies beinhaltet jedoch die Einschränkung, dass diese nur interpoliert werden kann, da die Frequenz in der Ortskurve auf keiner Achse aufgetragen ist. Deshalb werden die Frequenzen der Messpunkte vor und nach dem Nulldurchgang ausgelesen und je nach Abstandsverhältnis wird zwischen diesen beiden Frequenzen interpoliert. So kann die Frequenz zwar nicht genau bestimmt werden, erhält aber einen interpolierten Schätzwert, der andernfalls nicht zu bestimmen ist.

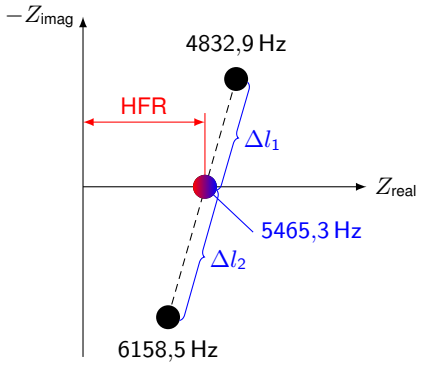

Abbildung 8.12: Ermittlung des ohmschen Hochfrequenzwiderstandes über den Schnittpunkt mit der Ordinate und Berechnung der Frequenz über lineare Interpolation zwischen den Frequenzen der beiden Messpunkte.

8.5.3 Hover Information

Die zuvor erwähnten interaktiven Inhalte, auch als Hover Information bezeichnet, stehen nur im .html Format zur Verfügung. Wird mit dem Mauszeiger ein Datenpunkt überfahren, so werden die hinterlegten Daten angezeigt, wie in Abbildung 8.13 für einen Messpunkt und den Hochfrequenzwiderstand exemplarisch dargestellt. Diese Funktionalität steht auch für alle Grafiken zur Verfügung.

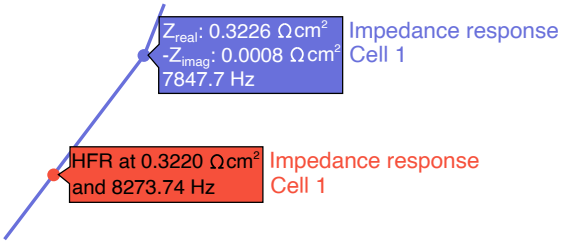

Abbildung 8.13: Darstellung der interaktiven Hoverinformation beim Überfahren der Datenpunkte mit dem Mauszeiger. Es wird in der Darstellung der Ortskurve der Real- sowie der Imaginärteil, die Nummer der Zelle und die jeweilige Signalfrequenz ausgegeben. Beim Hochfrequenzwiderstand wird der interpolierte Widerstand sowie die interpolierte Frequenz angezeigt.

Damit diese Daten bereitgestellt werden können, muss zuerst ein String mit Informationen für jeden Punkt erstellt werden. Da für die Werte nur die Koordinaten des jeweiligen Datenpunktes ausgelesen werden müssen, wird diese über die Notation %{x:.4f} gemacht. Dies kann in einen String eingebunden werden, wie es im Codeausschnitt 8.16 der Fall ist.

```
_text_Hover = "Z<sub>real</sub>: %{x:.4f} \ohm cm<sup>2</sup><br>
                        -Z<sub>imag</sub>: %{y:.4f} \ohm cm<sup>2</sup><br>"
_Hover_Frequency = string.(round.(_Frequencies, digits = 1)) .* " Hz"
_Line_Plot = PlotlyJS.scatter(; x = _Impedance_Cells_Real, y = _Impedance_Cells_Imag,
                        mode = "lines+markers", name = "Impedance response<br>Stack",
                        hovertemplate = _text_Hover .*  _Hover_Frequency)
```

Listing 8.16: Definieren der Hoverinformation mittels String.

Die Frequenzinformation wird über den Frequenzvektor eingelesen und mit der Einheit ergänzt. Über den Befehl hovertemplate werden die beiden Informationen zusammengefügt und sind so für jeden Messpunkt interaktiv in der Grafik verfügbar.

9 ERGEBNISSE

Für die Darstellung der Ergebnisse wird zunächst der Messaufbau sowie der Auswertealgorithmus validiert. Dies erfolgt zum einen mittels qualitativen Vergleichs von publizierten Messergebnissen und zum anderen durch eine Vergleichsmessung mit einem Industriegerät. Anschließend werden einige Ergebnisse von Messungen an verschiedenen Stacks bei unterschiedlichen Randbedingungen gezeigt und deren Auswirkungen interpretiert. Da es sich bei den Messungen um Kundenprojekte handelt, unterstehen diese einer Geheimhaltungsvereinbarung. Deshalb sind deren Daten in den Abbildungen ohne Zahlenwerte auf den Achsen aufgetragen und zeigen nur die jeweilige Einheit. Die Skalierung der beiden Achsen zueinander bleibt bei der Darstellung der Ortskurven jedoch erhalten, wodurch die Form der Ortskurve nicht verändert wird.

9.1 Messungen am Recycalyse-Stack

Die ersten Messungen dienten einem Funktionsnachweis für den Messaufbau und um dessen Spezifikationen zu garantieren. Deshalb basiert auch die Entwicklung der Algorithmen auf diesen Daten, womit diese das Fundament für die Auswertung darstellen. Das Messobjekt ist ein PEM-Stack, der aus dem europäischen Forschungs- und Innovationsprojekt Recycalyse von Horizon 2020 stammt. Der Stack besitzt 14 Zellen mit einer Zellfläche von $75\,\mathrm{cm}^2$. Gemessen wird im Betriebspunkt bei einem Gleichstromanteil von 100 A, einer eingangsseitigen Wassertemperatur von $50\,^\circ\mathrm{C}$ an der Anode und einem Kathodendruck von 5 bar. Die Amplitudenhöhe des Wechselstromsignales beträgt 10 A. In Abbildung 9.1 ist der Amplituden- sowie der Phasengang des Recycalyse-Stack ersichtlich.

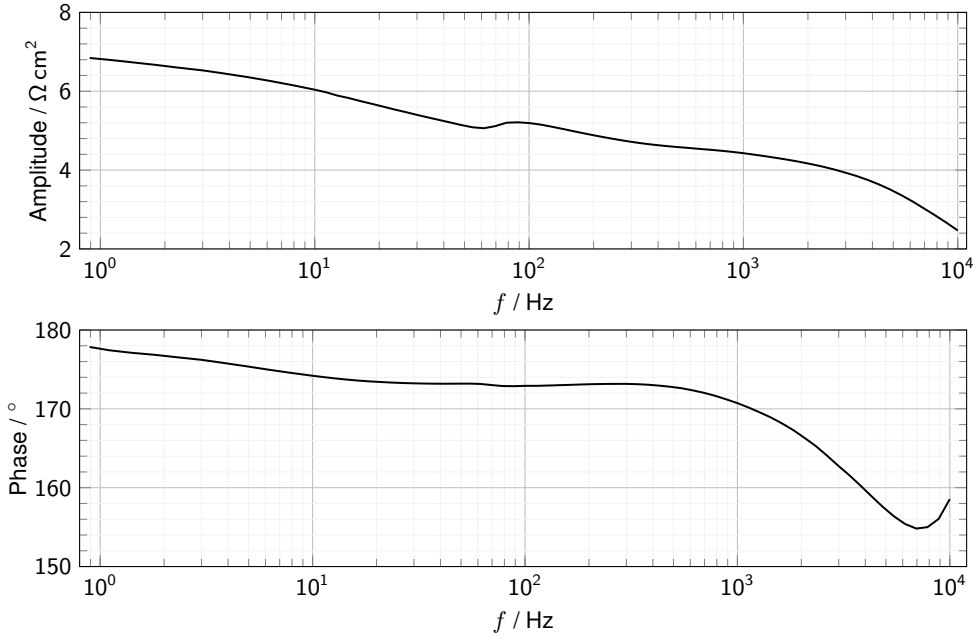

Abbildung 9.1: Amplituden- und Phasengang des Recycalyse Stack bis zu einer Frequenz von 10 kHz.

Die kapazitiven Eigenschaften des Stacks führen zu einer Abnahme des Phasenwinkels. Da die Induktivität mit zunehmender Frequenz dominant wird, steigt der Winkel in diesem Bereich wieder. Ist die Induktivität auch bei hohen Frequenzen größer als die kapazitiven Anteile, erhält man in der Ortskurve keinen Nulldurchgang. Da die kapazitive Impedanz mit steigender Frequenz kleiner wird, nimmt auch die Amplitude zunehmend ab.

Die Ortskurve des Stacks ist in Abbildung 9.2 illustriert und zeigt die beiden charakteristischen Halbkreiskurven. Wie die in diesem Kapitel noch folgenden Messungen zeigen werden, ist diese Kurve etwas untypisch, da sich der kleinere der beiden Halbkreise nicht im Hochfrequenten Messbereich und damit links im Diagramm befindet. In der Literatur findet man jedoch auch diese Kurvenform [85]. Außerdem ist im Kurvenverlauf eine Anomalie zu erkennen, die sich im Bereich oberhalb von $365\,\mathrm{m\Omega cm^2 n^{-1}}$ in Form einer Überschneidung der Daten zeigt. Der Grund hierfür ist ein systematischer Fehler im Messsystem. Diesem Fehler liegt ein Frequenzsprung in den Einprägefrequenzen zugrunde, der so nicht vorgesehen ist. Aufgrund dieses Frequenzsprunges bildet sich auch in den Ortskurven der Einzelzellen diese Anomalie aus, wie in Abbildung 9.3 ersichtlich. Der Hochfrequenzwiderstand liegt in einem für PEM-Elektrolyseure typischen Bereich, wenn man diesen mit der Literatur vergleicht [86].

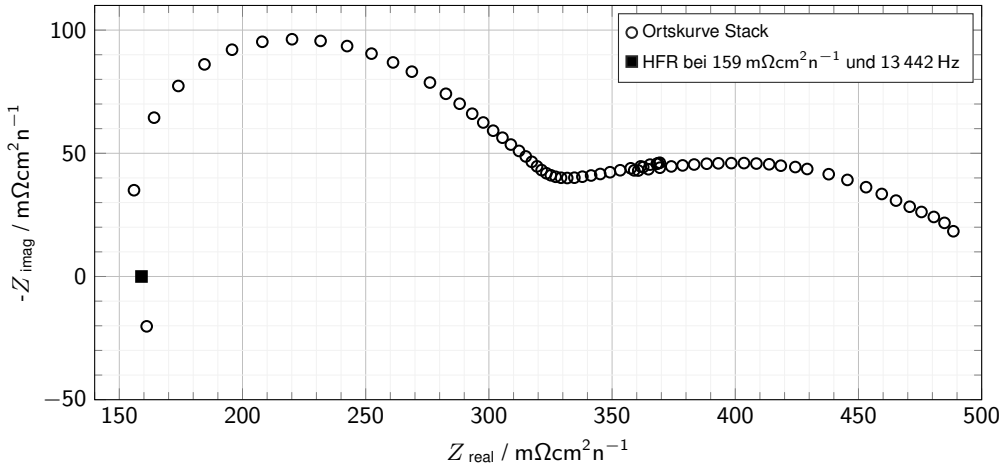

Abbildung 9.2: Ortskurve des Recycalyse Stack normiert auf eine Zelle und mit Hochfrequenzwiderstand.

Der Einprägefehler lässt sich auch im ungewöhnlichen Messverlauf der Amplitudengänge durch die Zunahme im Bereich von 600 Hz bis 900 Hz über alle Zellen erkennen, wie Abbildung 9.3 zeigt. Die Phasengänge der Zellen zeigt ebenfalls einen ungleichmäßigen Verlauf mit einer Spreizung von 5° bei niedriger Frequenz, die sich in den hohen Frequenzen auf 10° steigert. Außerdem kann man eine Annäherung im niederfrequenten Abschnitt des Grafen erkennen, die sich bei gering höherer Frequenz wieder auf den ursprünglichen Wert ausweitet. Die Anomalie in den Daten lässt sich in der Phase, wenn auch nur in abgeschwächter Form und nicht bei allen Zellen, erkennen. Im Gegensatz zur Amplitude stimmt der Phasenverlauf nicht mit der des Stacks aus Abbildung 9.1 überein. Die Phasen steigen bereits ab einem Frequenzwert von 100 Hz an, was auf den kleiner werdenden kapazitiven Blindwiderstand zurückzuführen ist.

85 Vgl. Suermann, Michel and Beard Patru, Alexandra and Schmidt, Thomas and Büchi, Felix 2017, S. 7.
86 Vgl. Suermann, Michel and Beard Patru, Alexandra and Schmidt, Thomas and Büchi, Felix 2017, S. 7.

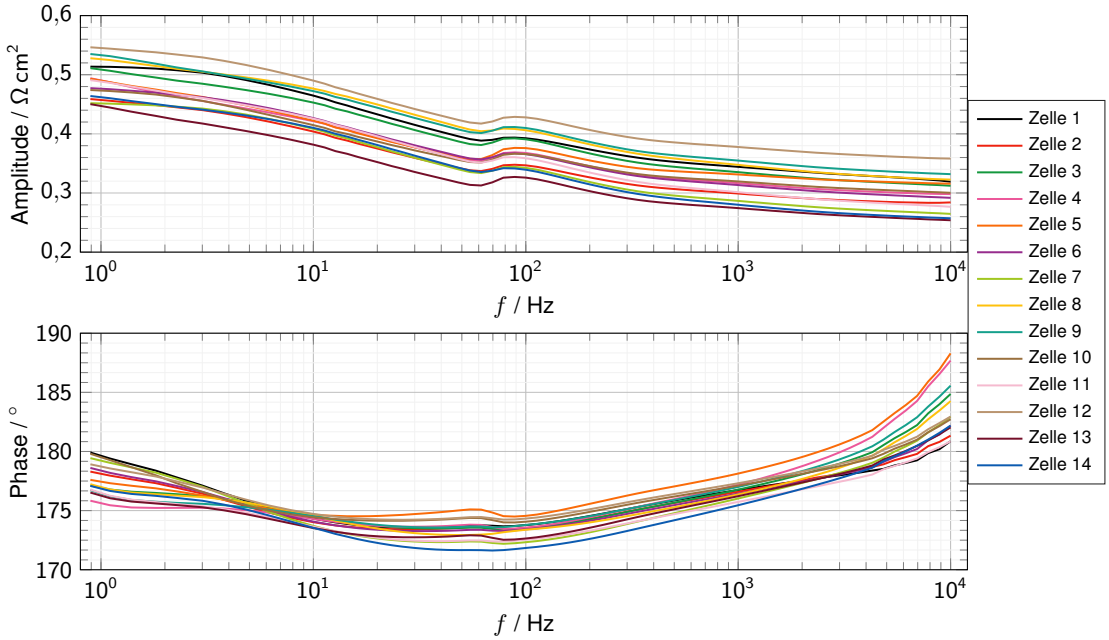

Abbildung 9.3: Amplituden- und Phasenplot der Einzelzellen des Recycalyse Stack.

Die Ortskurven der Einzelzellen sind in Abbildung 9.4 aufgetragen. Im Vergleich mit der Literatur[87], liegen diese im selben Wertebereich und auch der qualitative Vergleich der Kurvenform lässt auf eine valide Messung sowie Auswertung schließen. Auch die Anomalie der Impedanz findet sich in den Einzelzellen wieder.

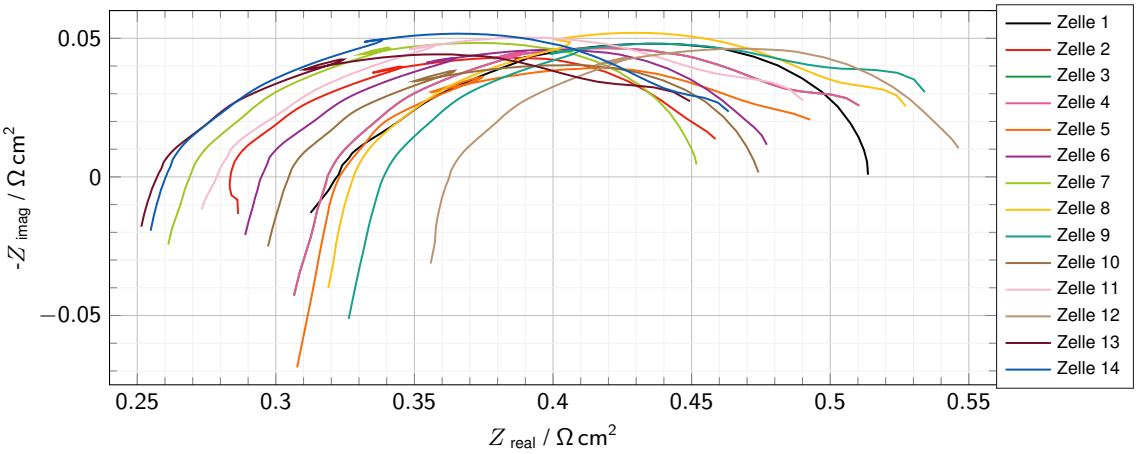

Abbildung 9.4: Ortskurven der Einzelzellen des Recycalyse Stack. Aufgrund der Vielzahl an Datenpunkten erfolgt die Darstellung in Linienform und nicht wie üblich der einzelnen Datenpunkte.

Wie homogen die Membranwiderstände über den Stack verteilt sind, lässt sich anhand des Hochfrequenzwiderstandes der einzelnen Zellen darstellen, wie Abbildung 9.5 verdeutlicht. Dabei handelt es sich um keinen gemessenen Wert, sondern einer auf Messdaten basierenden Interpolation. Die Grafik zeigt auch, dass der Widerstand der Membranen einer Schwankung von bis zu 100 $\mathrm{m\Omega\,cm^2}$ unterliegt, was sich wiederum mit der Streuung der Amplitudenverläufe aus Abbildung 9.3 deckt. Auch die Frequenz des Hochfrequenzwiderstandes kann abgeschätzt werden, jedoch mittels Interpolation, weshalb es sich auch hier

[87] Vgl. C. Immerz, B. Bensmann, P. Trinke, M. Suermann and R. Hanke-Rauschenbach 2018, S. 1297–1298.

um keinen gemessen Wert handelt. Bei der Frequenz ergibt sich eine hohe Schwankungsbreite von bis zu 6 kHz zwischen einzelnen Zellen. Diese Schwankung kann darauf zurückgeführt werden, dass die Frequenz der Einprägung logarithmisch gesteigert wird und damit die Frequenzschritte im hohen Frequenzbereich dementsprechend groß sind. Durch die lineare Interpolation zwischen diesen Frequenzschritten ergibt sich die dargestellte Unsicherheit der Werte.

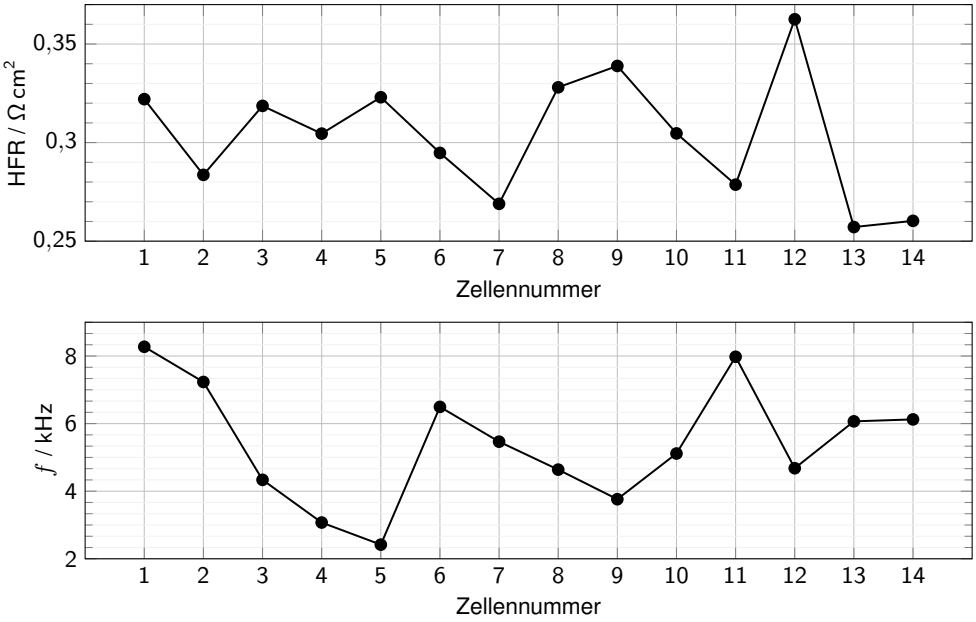

Abbildung 9.5: Hochfrequenzwiderstand und dessen Frequenz für alle Zellen des Recycalyse Stack.

Die insgesamt schlechte Performance des Stacks ist darauf zurückzuführen, dass sich der Katalysator an der Anode gelöst hat. Der Grund dafür ist das Bindematerial, welches den Katalysator mit der Elektrode verbindet. Sobald Spannung angelegt wurde, war das Bindematerial nicht mehr stabil und wurde zusammen mit Katalysator aus dem Stack gespült. Somit konnte keine Kontaktierung von der Elektrode zur Gasdiffusionsschicht hergestellt werden. Da nicht alle Zellen gleichermaßen betroffen sind, ergeben sich die Unterschiede zwischen den Zellen.

9.2 Validierung des Messaufbaues sowie des Auswertealgorithmus

Um den Messaufbau sowie die anschließenden Auswertungen auch empirisch validieren zu können, werden diese mit einem kommerziellen EIS-Messsystem verglichen. Das Vergleichsgerät ist ein Gamry Reference 3000 welches einen maximalen Strom von 30 A ausgeben und messen kann, weshalb auch bei einer niedrigen Stromdichte gemessen wird. Vermessen wurden die Zellen sowie der gesamte Stack. Der Verlauf der Stackmessung, in Abbildung 9.6 ersichtlich, der beiden Ortskurven lässt die Schlussfolgerung zu, dass die Messung und Auswertung qualitativ auf dem Niveau des Industriegerätes sind. Da die Daten der Geheimhaltung unterliegen, kann die Abweichung keinem Zahlenwert zugeordnet werden. Nur die Differenz der beiden ermittelten Hochfrequenzwiderstände gibt ein Indiz dafür, dass sich die Abweichungen im einstelligen mΩ Bereich bewegen. Aufgrund der Komplexität der Messkette, des Elektrolysesystems mit seinen Peripheriegeräten und dass die Messungen mit einem zeitlichen Versatz von mehreren Wochen stattgefunden haben, kann dieser geringe Fehler als Funktionsnachweis betrachtet werden.

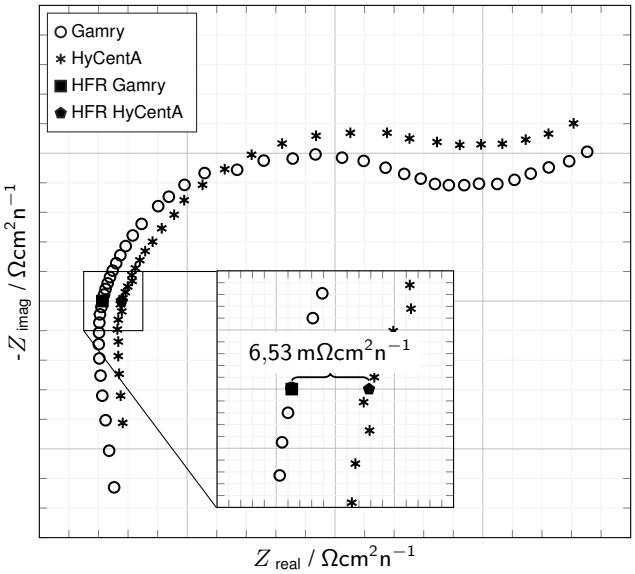

Abbildung 9.6: Validierung des HyCentA Messaufbaues sowie der Auswertung durch Gegenüberstellung zweier gemessener Ortskurve mit demselben Stack. Die Daten sind auf die Zellfläche sowie die Zellanzahl normiert.

9.3 Messungen am H-TEC Stack

Der zweite Stack der vermessen wurde, kommt von der Firma H-TEC, mit einer Zellfläche von $30\,\mathrm{cm}^2$ und 10 Zellen. Der Gleichstromanteil der Messung beträgt 60 A und die aufgebrachte Wechselstromamplitude erreicht einen Wert von 1 A bei einer Betriebsmitteltemperatur von $60\,^{\circ}\mathrm{C}$. Es handelt sich dabei um einen kompakten Stack, was sich auch in der Zugänglichkeit der Anschlüsse widerspiegelt. Daraus ergeben sich Kontaktierungsprobleme an den Anschlüssen, die sich vor allem in den Ortskurven sehr deutlich zeigen. Außerdem werden von den 10 Zellen nur 8 gezeigt, da an den Zellen 5 und 6 nur ein Netzbrummen von 50 Hz gemessen wurde.

Die schlechte Qualität der Messung ist in der Welligkeit der gemessenen Amplitudengänge ersichtlich, wie sie Abbildung 9.7 zeigt. Diese Welligkeit begrenzt sich jedoch auf einen Bereich von circa $15\,\mathrm{m\Omega\,cm}^2$ und verhält sich im Vergleich der Zellen annähernd konstant. Darüber hinaus beträgt die Differenz zwischen den Amplituden der Zellen bei niedriger Frequenz circa $150\,\mathrm{m\Omega\,cm}^2$ und nimmt mit zunehmender Frequenz auf $100\,\mathrm{m\Omega\,cm}^2$ ab. Diese Unterschiede ergeben sich aus der Abweichung von Zelle 2 und 4 nach unten, sowie Zelle 10 nach oben. Alle anderen Zellen verteilen sich über einen Bereich von wenigen $\mathrm{m\Omega\,cm}^2$. Die Kontaktierungsprobleme sind im Phasengang der Zellen so gut wie nicht sichtbar und auch deren Verlauf kann bis zu einer Frequenz von 1 kHz als deckungsgleich bezeichnet werden. Besonders der Vergleich mit dem Recycalyse-Stack in Abbildung 9.3 zeigt, dass es sich um einen sehr homogenen Verlauf aller Zellen handelt. Außerdem zeigt sich, dass die Hälfte der Phasenwinkel der Zellen im hohen Frequenzbereich zunimmt und die andere Hälfte abnimmt, was so beim Recycalyse-Stack nicht der Fall war und sich im Amplitudenverlauf auch nicht widerspiegelt beziehungsweise nicht stringent nachvollziehen lässt. Zelle 4 weist die zweitniedrigsten Amplitudenwerte auf und die größte positive Phasenverschiebung, wobei Zelle 10 die höchsten Amplitudenwerte bei der zweitgrößten positiven Phasenverschiebung zeigt. Auch der Vergleich der anderen Zellen lässt, zum jetzigen Zeitpunkt, kein nachvollziehbares Muster erkennen. Da der Phasenwinkel bei einigen Zellen abnimmt, dominiert hier der kapazitive und bei den anderen der induktive Anteil des Membran Elektroden Aufbaues, wobei die Kontaktprobleme ein eher kapazitives Verhalten begünstigen sollten.

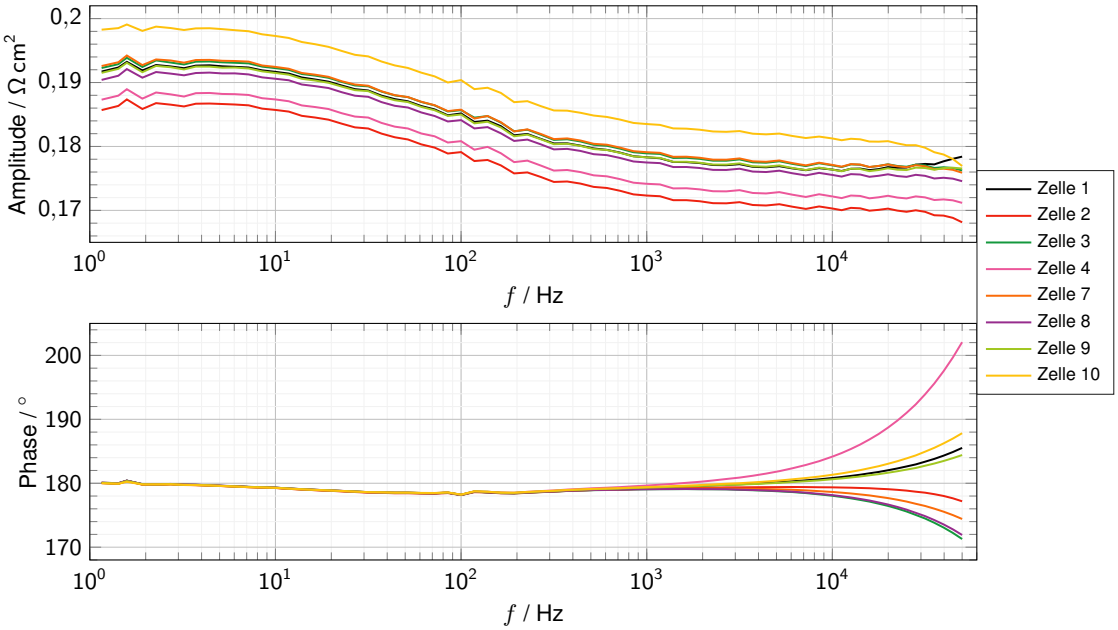

Abbildung 9.7: Amplituden und Phasen Plot der Einzelzellen des H-TEC Stack bei 60 °C.

Die Kapazität und Induktivität lässt sich in der Ortskurve noch besser erkennen, da ohne induktiven Anteil kein Nulldurchgang im Hochfrequenzbereich zustande kommt, wie in Abschnitt 5.3 bereits erläutert wurde. Abbildung 9.8 verdeutlicht, dass nur die Zellen mit positivem und damit induktiven Phasenverlauf einen Nulldurchgang besitzen.

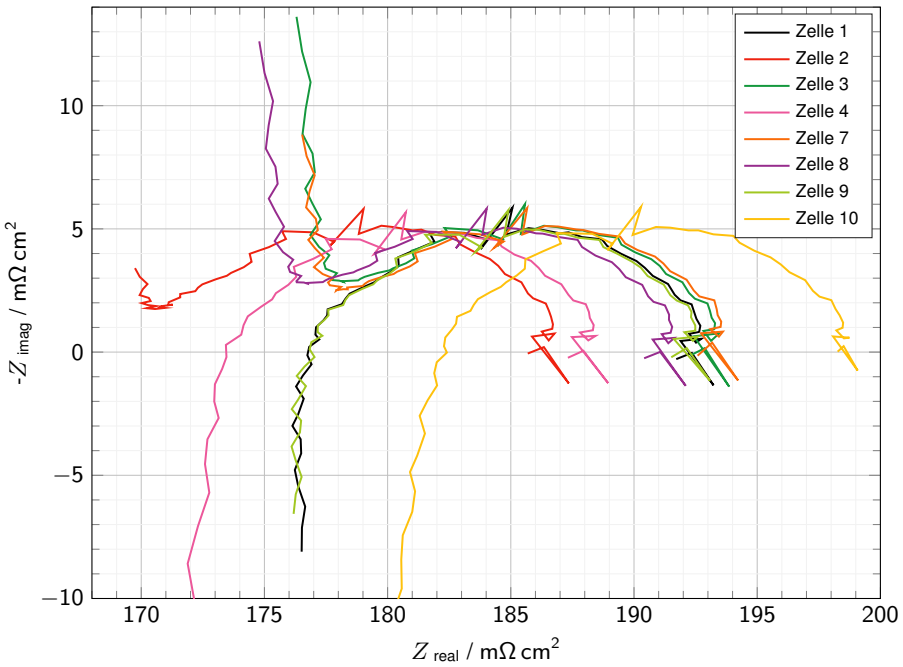

Abbildung 9.8: Ortskurve der Einzelzellen der Einzelzellen des H-TEC Stack bei 60 °C und einer maximalen Frequenz von 15 kHz.

Alle anderen Zellen ändern ihre Krümmung und damit ihren Verlauf hin zu höheren Imaginärteilen was auf die kapazitiven Anteile zurückzuführen ist. Da mit steigender Frequenz nach Gleichung (5.4) die Impedanz immer weiter abnehmen muss, müsste sich der kapazitive Blindwiderstand analog zur Frequenz verringern,

um einen nahezu vertikalen Verlauf zu erzeugen, weshalb angenommen werden kann, dass es sich hier um einen Messfehler handelt. Die schlechte Messqualität spiegelt sich auch in allen Zellen anhand der Welligkeit des Verlaufes wider. Da nur vier Zellen eine nachvollziehbare Ortskurve ausbilden und zwei davon einen nahezu identen Verlauf aufweisen, werden in weiterer Folge nur drei Zellen weiter betrachtet.

Ein weiterer Indikator für die schlechte Kontaktierung bei den Messungen sind die Ortskurven der Zellen 1 und 4 in Abbildung 9.9. Hierbei hat Zelle 1 beim Betriebszustand von 70 °C keinen Nulldurchgang, wie die Messungen bei den niedrigeren Temperaturen. Ein ähnlicher Messfehler tritt auch an Zelle 4 auf, nur das dieser hier bei der niedrigsten Temperatur von 50 °C auftritt.

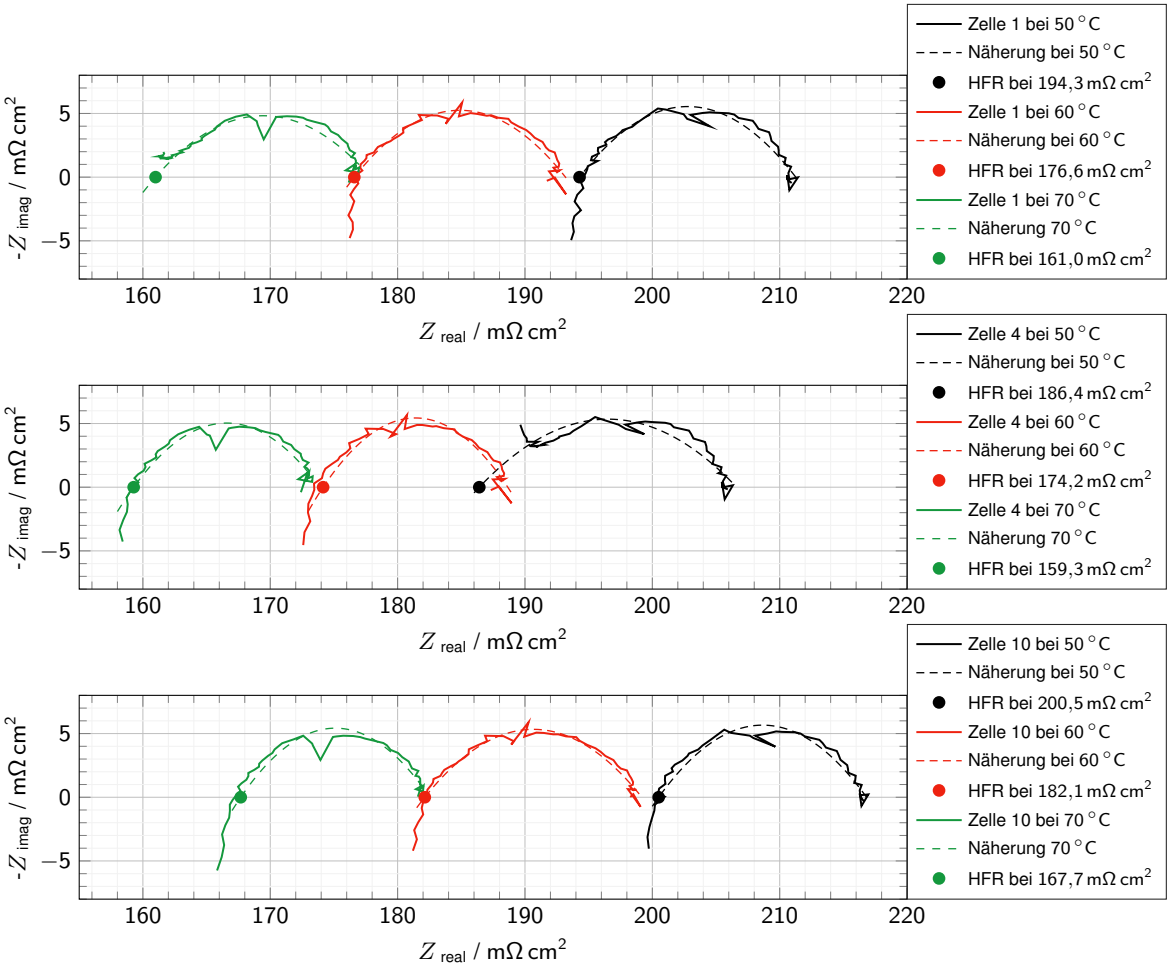

Abbildung 9.9: Ortskurve der Zelle, 1, 4 und 10 des H-TEC Stack bei 50 °C, 60 °C und 70 °C.

Aufgrund der schlechten Datenqualität werden die Ortskurven deshalb mit einem Polynom zweiten Grades angenähert. Somit ist es möglich, zumindest einen qualitativen Vergleich über den Einfluss einer Temperaturänderung des Betriebsmittels durchzuführen. Die unterschiedliche Güte der einzelnen Zellen spiegelt sich auch hier wider, indem sich die Ortskurven insgesamt entlang der Abszisse verschieben. Die Hochfrequenzwiderstände und deren absolute Änderung aufgrund der Temperaturdifferenz des Betriebsmittels sind in Tabelle 9.1 aufgelistet. Die Zellen sowie die Näherung mittels Polynoms sind in Abbildung 9.9 ersichtlich. Was alle drei Vergleiche zeigen ist, dass sich die Ortskurve und damit der Hochfrequenzwiderstand mit der Zunahme der Temperatur verringert. Durch die Zunahme der Protonenleitfähigkeit der Membran bei steigender Temperatur, verringert sich deren elektrischer Widerstand[88]. Was sich in Diagrammen rein

[88] Vgl. Ömer Faruk Selamet, Fatih Becerikli, Mahmut D. Mat, Yüksel Kaplan 2011, S. 11485.

qualitativ feststellen lässt, ist die annähernd lineare Abnahme des Widerstandes in dem betrachteten Temperaturbereich. Diese zeigt sich in der Verschiebung der Ortskurve auf der Abszisse und auch die Werte in Tabelle 9.1 untermauern diese These. Da auch die Durchmesser sowie die höchsten Werte auf der Ordinate über die Temperatur relativ konstant bleiben, hat diese nur einen Einfluss auf die Membran, nicht jedoch auf die Elektroden sowie Katalysatoren der Zelle.

	50 °C / mΩ cm^2	60 °C / mΩ cm^2	70 °C / mΩ cm^2	50 °C - 60 °C / mΩ cm^2	60 °C - 70 °C / mΩ cm^2
Zelle 1	194,3	176,6	161,0	17,7	15,6
Zelle 4	186,4	174,2	159,3	12,2	14,9
Zelle 10	200,5	182,1	167,7	18,4	14,4

Tabelle 9.1: Hochfrequenzwiderstände der Zellen 1, 4 und 10 bei Temperaturen des Betriebsmittels von 50 °C, 60 °C und 70 °C.

9.4 Messungen am Kundenstack bei verschiedenen Betriebspunkten

Laut den Vorgaben der HyCentA Research GmbH muss jeder Stack bevor er im Prüfstand getestet wird einem Drucktest unterzogen werden. Dies dient zum einen der Kontrolle möglicher Montagefehler und soll zum anderen die Bildung einer explosionsfähigen Atmosphäre in der Prüfzelle verhindern. Der Drucktest wird mit Stickstoff und anschließend mit Wasserstoff bei erhöhter Temperatur durchgeführt. Zusätzlich wird gemessen, wie viel Wasser durch die Membran bei verschiedenen Drücken von der Anode zur Kathode durchdiffundiert. Erst dann kann der Stack in den Prüfstand eingebaut und getestet werden.

Die unterschiedlichen Betriebspunkte ergeben sich aus der Änderung des Kathodendrucks, sowie dem Moment mit dem die Stackschrauben angezogen werden. Grundsätzlich ist die Erhöhung des Anzugsmoment kein üblicher Parameter beim Testen von Stacks. Dieses wurde erhöht, da während der Versuchsreihen, trotz vorherigem Drucktest, Leckagen auftraten.

9.4.1 Messungen mit unterschiedlichen Drücken

Die ersten drei Messungen erfolgten bei konstantem Anzugsmoment sowie konstanter Wassertemperatur von 75 °C am Anodeneingang. Der Amplituden- und Phasengang der ersten Messung sind in Abbildung 9.10 ersichtlich. Diese wurde nach dem Drucktest und einer Konditionierungsphase durchgeführt. Dadurch wird der Zustand des Stacks vor den Versuchsreihen dokumentiert und eine erste Aussage über die Güte und Leistungsfähigkeit kann getroffen werden. Aufgrund der qualitativen Darstellung ist die Einschätzung der Diagramme schwierig, weshalb der Vergleich über Größenordnungen zu üblichen Messwerten gezogen wird. Die Amplitudenwerte der Zelle 3 liegen in einem für PEM-Elektrolyseure gängigen Bereich. Die Zellen 1, 2 und 5 weichen davon ab, da die Spreizung um das Zehnfache höher ist als bei vergleichbaren Messungen. Dies lässt schon erahnen, dass die Zellen 4 und 6 weit außerhalb des angestrebten Bereiches liegen. Aufgrund der Skalierung wirkt der Verlauf von Zelle 3 annähernd linear, dieser nimmt jedoch mit zunehmender Frequenz ab und erreicht am Ende des Spektrums nur noch 35 % seines Startwertes. Da diese Abnahme linear erscheint, wird deutlich wie weit die beiden Zellen 4 und 6 vom Sollbereich entfernt sind. Auch der Phasengang zeigt auf den ersten Blick keine homogene Verteilung der einzelnen Zellen. Besonders bei niedrigen Frequenzen ergibt sich eine große Streuung, die weit über der normalen Schwankungsbreite liegt. Erst im letzten Drittel des Frequenzbandes zeigt sich ein charakteristisches Verhalten, indem sich die Phasen annähern und die Induktivität dominiert, was sich im Anstieg des

Phasenwinkels zeigt. Dies gilt jedoch nicht für die Zellen 2 und 3, da diese diesen Anstieg nicht aufweisen. Auch hier zeigen Zelle 4 und 6 insgesamt die größten Abweichungen.

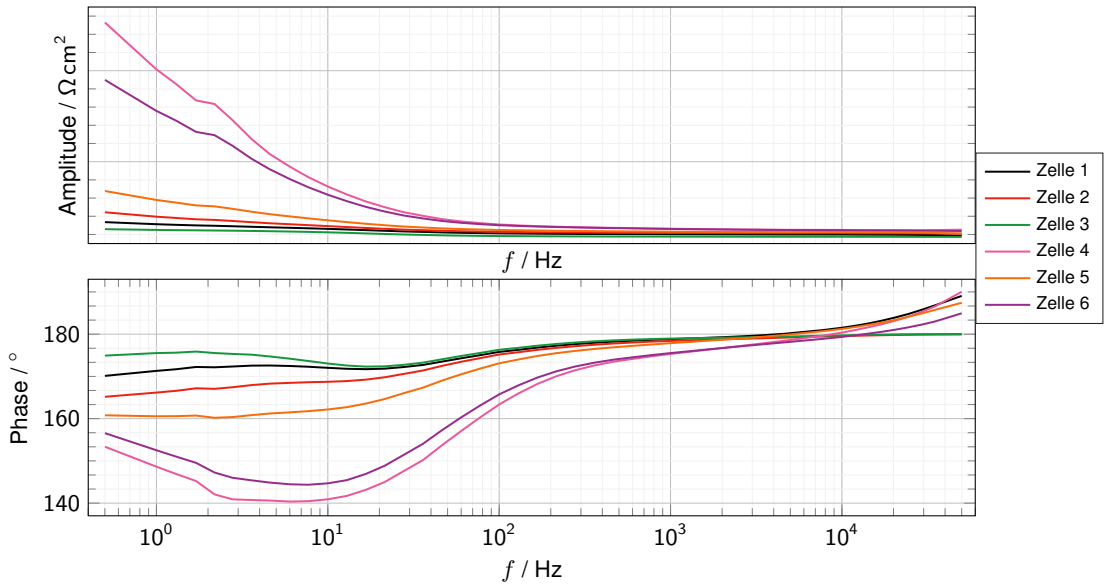

Abbildung 9.10: Amplituden- und Phasengang des Kundenstack bei der Eingangsmessung.

Was die Divergenz zwischen den Zellen noch deutlicher macht, sind die Ortskurven in Abbildung 9.11. Hier wird perspektivisch gezeigt, wie groß die Schwankungsbreite zwischen den Zellen ist. Um diese ins Verhältnis zu setzen, sind die Zellen 1 und 3 vergrößert dargestellt, da sich die beiden im zu erwartenden Bereich ausbilden. Die schlechte Erkennbarkeit im Zellvergleich, unterstreicht die ungenügende Qualität sowie die Inhomogenität des Stack in seinem Ausgangszustand. Wie in der Vergrößerung zu sehen, besitzen nicht alle Zellen einen Nulldurchgang, was mit der fehlenden Zunahme des Phasenwinkels der jeweiligen Zellen in Abbildung 9.10 korreliert. Da sich aber alle Zellen im hochfrequenten Abschnitt des Nyquist Plot in dem zu erwartenden Bereich befinden, lässt auf intakte Zellmembranen schließen und diese als Fehlerquellen ausschließen.

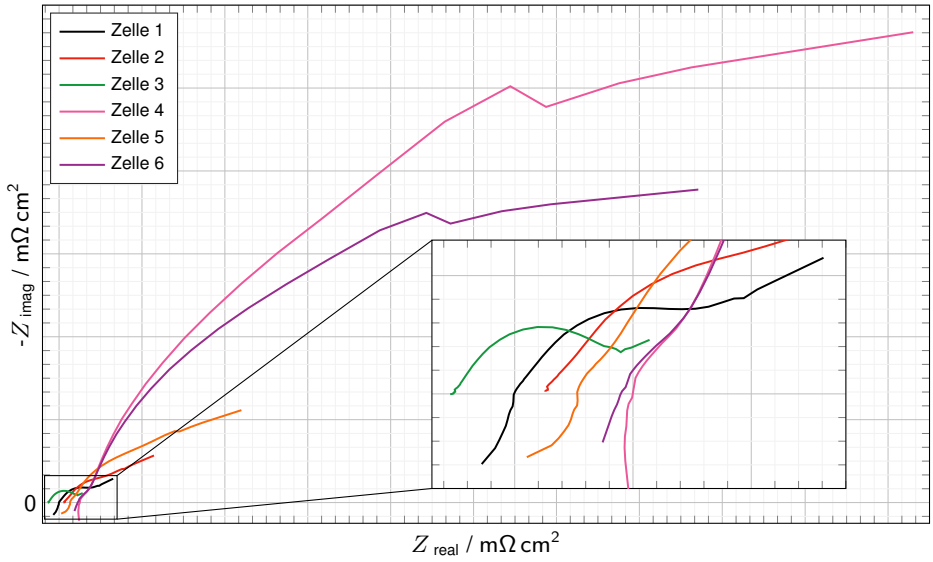

Abbildung 9.11: Ortskurven der Zellen bei der Eingangsmessung. Aufgrund der großen Spreizung zwischen den Zellen wird Zelle 1 und 3 vergrößert dargestellt.

Bei der zweiten Messung wird der Kathodendruck um 60 % erhöht. Dies hat in diesem Fall zur Folge, dass sich die Amplituden weiter annähern, jedoch immer noch nicht im üblichen Wertegebiet bewegen. Analog dazu verbessert sich auch der Phasengang, wobei die Hälfte der Zellen im hochfrequenten Bereich ein kapazitives Verhalten zeigt. Der Amplituden- und Phasengang ist in Abbildung A.1 gemeinsam mit der dazugehörigen Ortskurve in Abbildung A.2 im Anhang illustriert. Die Ortskurven lassen im perspektivischen Vergleich erahnen, dass sich Werte verbessert haben, die Spreizung zwischen den Zellen aber immer noch viel zu groß ist. Außerdem lässt sich im hochfrequenten Bereich erkennen, dass sich die drei Zellen mit den kapazitiven Anteilen auch hier durch die Veränderung der Krümmung hervorheben. Diese Messung festigt die Annahme, dass die schlechten Ergebnisse nicht auf die Membran zurückzuführen sind, da sich die Werte bei hohen Frequenzen im erwarteten Bereich bewegen.

Da bei der vorangegangenen Messung eine Leckage am Stack festgestellt wurde, wird der Druck in der dritten Messung um 22 % des Ausgangswertes reduziert. Auch bei diesem reduzierten Druck stellt sich im Vergleich zur Ausgangsmessung eine Verbesserung ein, wie in Abbildung 9.12 ersichtlich. Dies wird deutlich, wenn man die Abstände der Amplituden zueinander vergleicht, da die Spreizung der untersten vier Zellen nahezu gleichgeblieben ist. Auch der Phasengang zeigt eine Reduzierung der Spreizung zwischen den Phasenwinkeln. Außerdem sticht der untypische Verlauf der Phasen im hohen Frequenzbereich heraus. Statt ein induktives Verhalten und damit eine Zunahme des Phasenwinkels, zeigt sich durch die Abnahme ein kapazitives Verhalten von allen Zellen. Im Gegensatz zum synchronen Verlauf bei niedrigen Frequenzen, zeigen die Zellen im hochfrequenten Bereich unterschiedliche Ausprägungen. War Zelle 1 bisher immer eine der besten, so zeigt diese bei dieser Messung ein stark kapazitives Verhalten. Im Gegensatz dazu zeigt Zelle 4, welche bisher nur durch ihre schlechten Eigenschaften aufgefallen ist, eine viel geringeres kapazitives Verhalten.

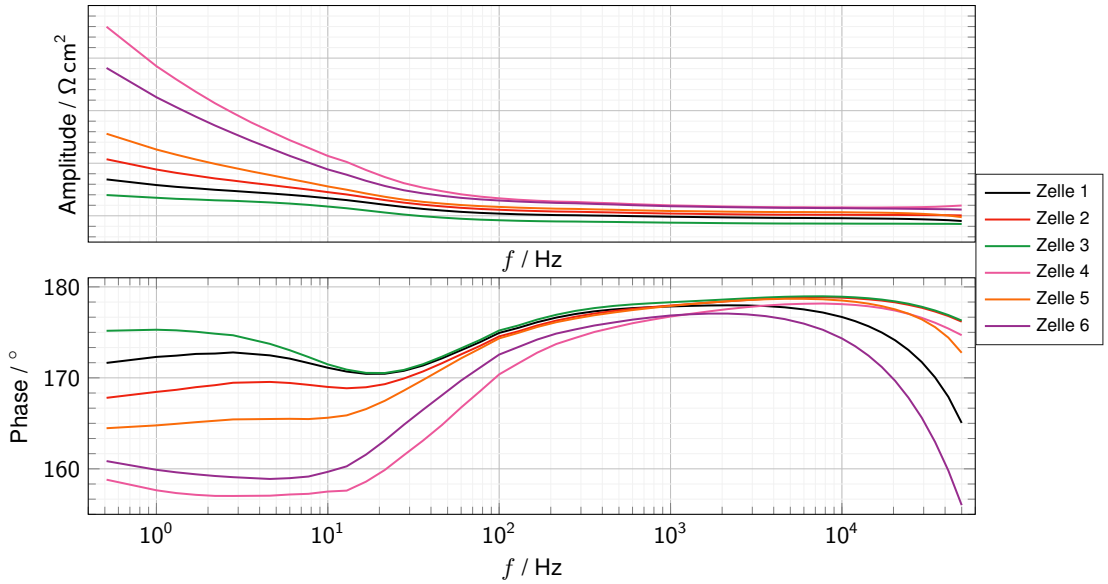

Abbildung 9.12: Amplituden- und Phasengang der dritten Messung mit einem im Vergleich zur Eingangsmessung um 22 % verringerten Druck.

Das im Vergleich zur Eingangsmessung verbesserte Verhalten lässt sich wiederum besonders gut anhand der Ortskurven festmachen, wie Abbildung 9.13 illustriert. Die Zellen zeigen immer noch eine erhebliche Differenz in ihrer Ausprägung, jedoch haben die Unterschiede abgenommen. Was sich auch hier wieder feststellen lässt ist der Messfehler, der sich auf alle Zellen gleichermaßen auswirkt. Was sich im Phasengang schon angedeutet hat, zeigt sich an der Ortskurven deutlich, da keine mehr einen Nulldurchgang

aufweist. Die Zellen bewegen sich im Bereich mit hoher Frequenz auf der Ordinate wieder nach oben in den kapazitiven Quadranten. Dabei sind die Phasenausschläge aus Abbildung 9.12 kongruent mit der neuerlichen Zunahme des imaginären Anteils der Impedanz. Zu diesem Zeitpunkt ist noch nicht klar, ob diese Kapazität in den Zellen aufgebaut wird oder ob die Kabel der Leistungsversorgung, aufgrund der örtlichen Nähe, einen Einfluss auf die Messleitungen haben.

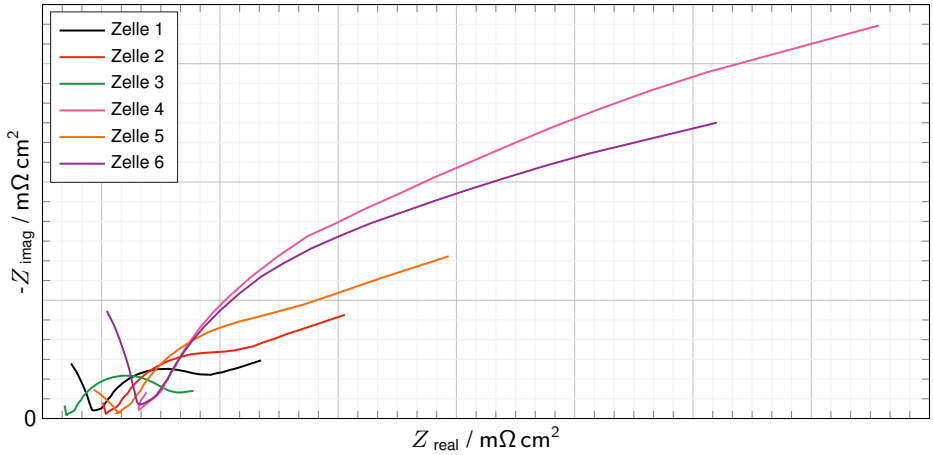

Abbildung 9.13: Ortskurven der dritten Messung mit einem im Vergleich zur Eingangsmessung um 22 % verringerten Druck.

9.4.2 Messungen mit erhöhtem Anzugsmoment der Stackschrauben

Im nächsten Schritt wird das Anzugsmoment der Stackschrauben um 10 Nm erhöht, um einer erneuten Leckage vorzubeugen. Außerdem liegt der Kathodendruck wieder bei 22 % des Ausgangsdruckes aus Messung 1. Ein höherer Anpressdruck sollte die Kontaktfläche erhöhen und damit die Impedanz verringern. Wie die neuerliche Verringerung der Spreizung zwischen den Amplituden in Abbildung 9.14 zeigt, ist dies auch der Fall.

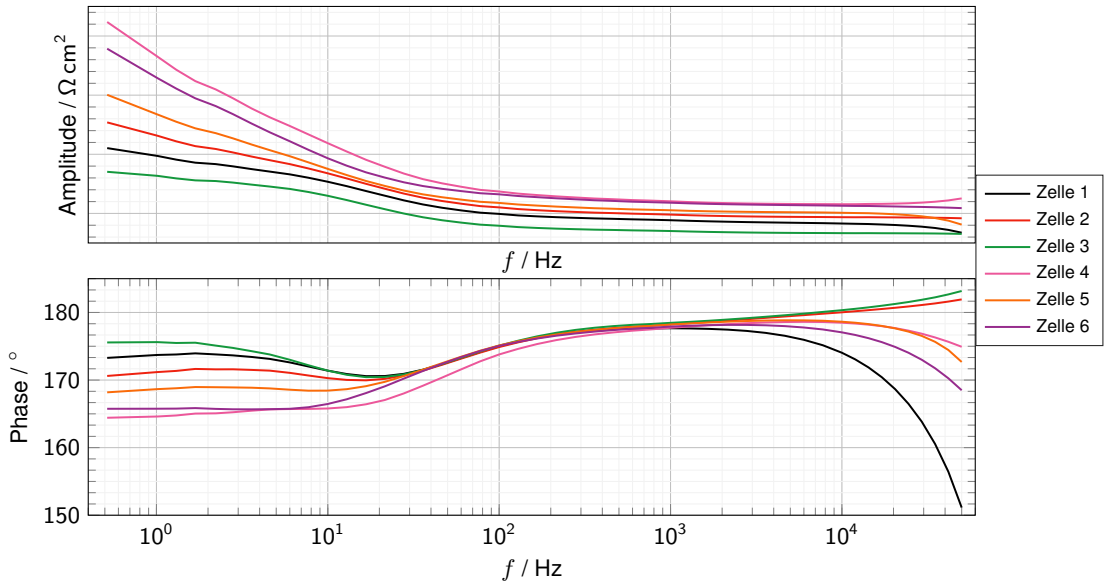

Abbildung 9.14: Amplituden- und Phasengang der dritten Messung mit einem 10 Nm höheren Anzugsmoment.

Zu diesem Zeitpunkt bewegen sich die gemessenen Werte zum ersten Mal im erwarteten Bereich. Auch die

Phasen ergeben ein homogeneres Bild und weichen nur noch bei niedrigen und hohen Frequenzen voneinander ab. Was im Vergleich zur vorhergehenden Messung in Abbildung 9.12 auffällt, ist die Zunahme der Kapazität in Zelle 1, die sich durch das Abfallen des Phasenwinkels zeigt. Im Gegensatz dazu nähert sich die vormals schlechteste Zelle 6 den anderen an, was mit einer Reduzierung der Kapazität gleichzusetzen ist. Auch in diesem Fall verdeutlicht die Ortskurve wieder, dass ein Nulldurchgang nur bei zunehmendem Phasenwinkel erreicht wird, wie die Ortskurven der Zellen 2 und 3 in Abbildung 9.15 unterstreichen. Auch die Ortskurven bewegen sich jetzt in einem Wertebereich, der mit anderen Stacks vergleichbar ist. Trotzdem zeigen die großen Unterschiede innerhalb der Ortskurven, dass der Stack große Schwankungen zwischen den einzelnen Zellen aufweist. Auch hier ist fraglich ob die starke Zunahme des Imaginärteils im hochfrequenten Bereich bei manchen Zellen aus der Zelle selbst ober von der Zuleitung des Stacks kommt.

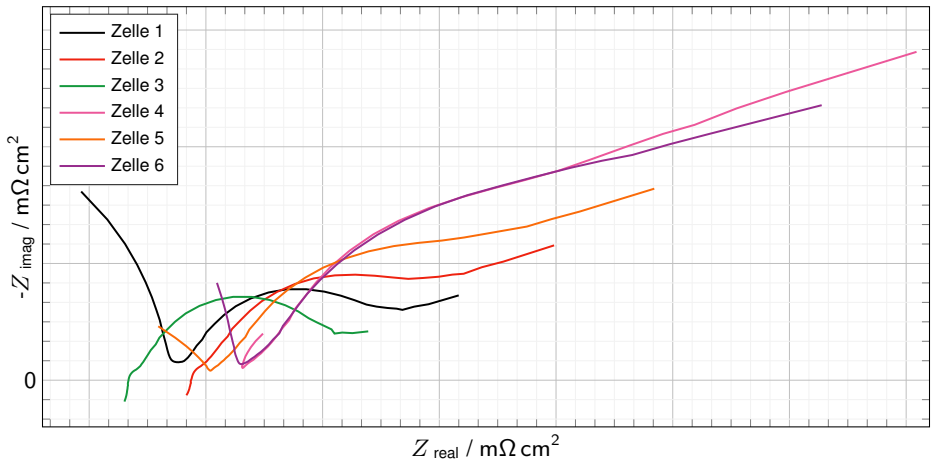

Abbildung 9.15: Ortskurven der dritten Messung mit einem 10 Nm höheren Anzugsmoment.

Als nächstes wird der Druck wieder auf den Wert der Ausgangsmessung angehoben. Wie der Amplitudengang in Abbildung 9.16 zeigt, hat dies keinen merklichen Einfluss auf deren Verlauf.

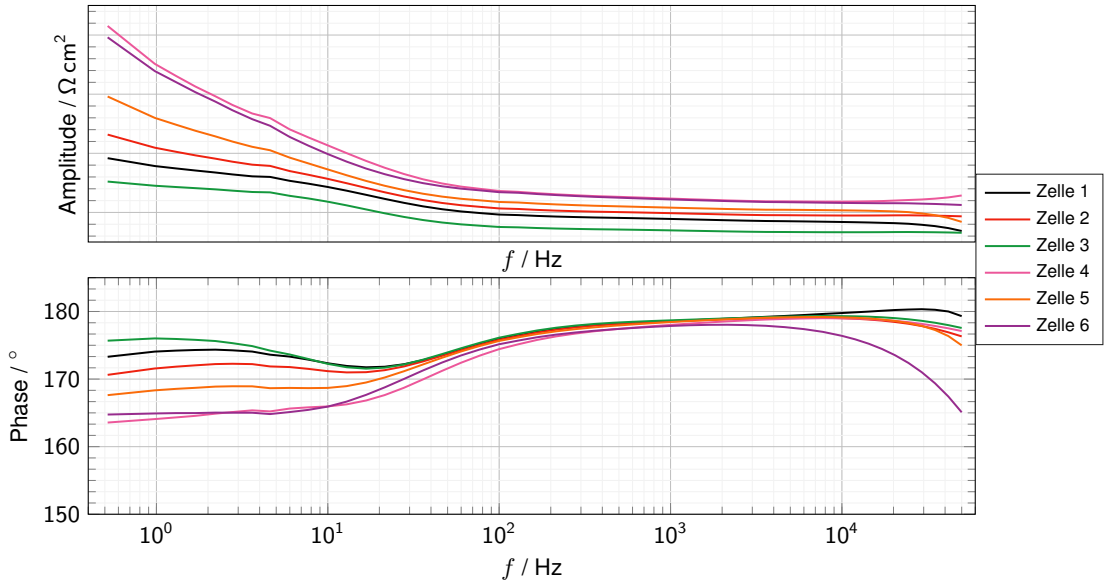

Abbildung 9.16: Amplituden- und Phasengang der vierten Messung mit einem 10 Nm höheren Anzugsmoment.

Nur Zelle 6 zeigt eine höhere Impedanz bei niedrigen Frequenzen im Vergleich zur vorherigen Messung.

Im Gegensatz dazu ändert sich im Phasengang bei niedrigen Frequenzen wenig, nur Zelle 4 nähert sich den anderen Zellen in deren Verlauf an. Den markantesten Unterschied weist jedoch Zelle 1 bei hohen Frequenzen auf. Innerhalb dieser Messreihe entwickelt sich diese von der Zelle mit der höchsten Kapazität zur Zelle mit der geringsten. Auch Zelle 2 und 3 zeigen bei dieser Messung wieder ein kapazitives Verhalten.

Im niederfrequenten Abschnitt der Ortskurven ergeben sich nur marginale Änderungen. Da bei der Messung wieder alle Phasenwinkel in den negativen Wertebereich abdriften, ergibt sich auch kein Nulldurchgang. Zelle 1 hat jedoch zwei Nulldurchgänge, was bei allen bisherigen Messungen ein Unikum darstellt. Dies lässt sich auch aus dem Phasengang der Zelle ablesen, da dieser zuerst ansteigt und nur in den letzten Frequenzen wieder abfällt und dem allgemeinen Trend folgt.

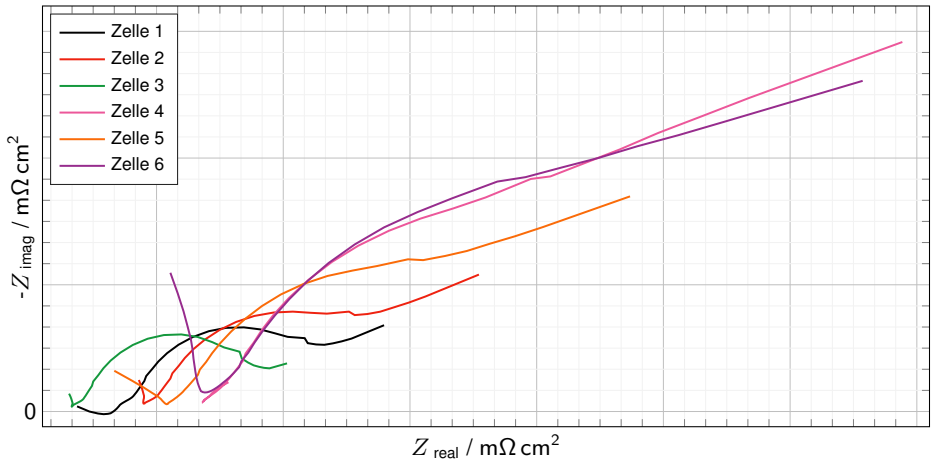

Abbildung 9.17: Ortskurven der vierten Messung mit einem 10 Nm höheren Anzugsmoment.

Da sich keine Änderung der Betriebszustände stringent dem Zellverhalten zuordnen lässt, können keine allgemeinen Aussagen über die Güte des Prüflings getroffen werden. Einige Verbesserungen beziehungsweise Verschlechterungen waren nur lokal bei einer Messung feststellbar. Es zeigt sich nur, dass sich mit jeder neuen Messung die Stackeigenschaften verbesserten und sich über die Zellen ein insgesamt homogeneres Verhalten ergibt. Da sich die Tests im jeweiligen Betriebspunkt nicht nur auf die Impedanzspektroskopie beschränken, wird der Stack für einen längeren Zeitraum auf diesem Betriebspunkt gehalten. Dadurch ergeben sich insgesamt mehrere Stunden an Versuchszeit. Da sich die Nulldurchgänge, sofern vorhanden, immer im zu erwartenden Bereich befanden, kann die Membran als Fehlerquelle so gut wie ausgeschlossen werden. Aufgrund der großen Schwankungen innerhalb der Zellen und dem Verlauf der Ortskurven, liegt das Problem wahrscheinlich beim Elektrolyten in den Elektroden. Diese benötigen eine Konditionierungsphase und unterliegen auch Degradationserscheinungen und sind auch maßgeblich für die Durchmesser der Halbkreise verantwortlich, die sich über die Testdauer stark verändert haben. Auch eine Vergrößerung der elektrochemisch aktiven Fläche ist denkbar. Aus diesem Grund wurden mehrstündige Versuchsreihen durchgeführt, bei denen zuerst ein konstanter Strom und anschließend eine konstante Spannung gehalten wurde.

9.4.3 Messung nach der erneuten Konditionierungsphase

Nach einer zweitägigen Konditionierungsphase wurde eine finale Impedanzmessung durchgeführt. Während dieser Phase wurde der Stack mit einem konstanten Strom betrieben. Dadurch konnte eine kontinuierliche Verbesserung, ersichtlich in der Zunahme der Stromdichte, festgestellt werden. Aufgrund der limitierten Testzeit konnten die zuvor durchgeführten Tests nicht wiederholt werden.

Die Amplituden sowie die Phasen dieser finalen Messung sind in Abbildung 9.18 ersichtlich. Im Vergleich zur ersten Messung in Abbildung 9.10, zeigt sich eine um ein vielfaches homogenere Verteilung der einzelnen Zellen. Auch die Spreizung der Amplitude bleibt über das gesamte Frequenzband, bis auf ein paar Ausreißer in den höchsten Frequenzen, annähernd gleich. Auch bei den Phasen zeigt sich ein analoges Bild. Die Spreizung in den niedrigen Frequenzen hat sich verringert, nur Zelle 6 weist einen etwas unterschiedlichen Verlauf auf. Der homogene Phasengang erstreckt sich jedoch nicht über das gesamte Frequenzband. Das Verhalten in dem für den Hochfrequenzwiderstand wichtigen Frequenzbereich unterscheidet sich wieder deutlich in der Ausprägung der Phasenverläufe. Zelle 2 und 3 zeigen ein stark induktives Verhalten und auch Zelle 5 besitzt dominante induktive Anteile. Zelle 4 und 6 stagnieren beziehungsweise weisen nur kapazitive Tendenzen auf. Zelle 1 zeigt zwar zu Beginn des hochfrequenten Bereiches ein kapazitives Verhalten, dieses kehrt sich aber in weiterer Folge wieder um.

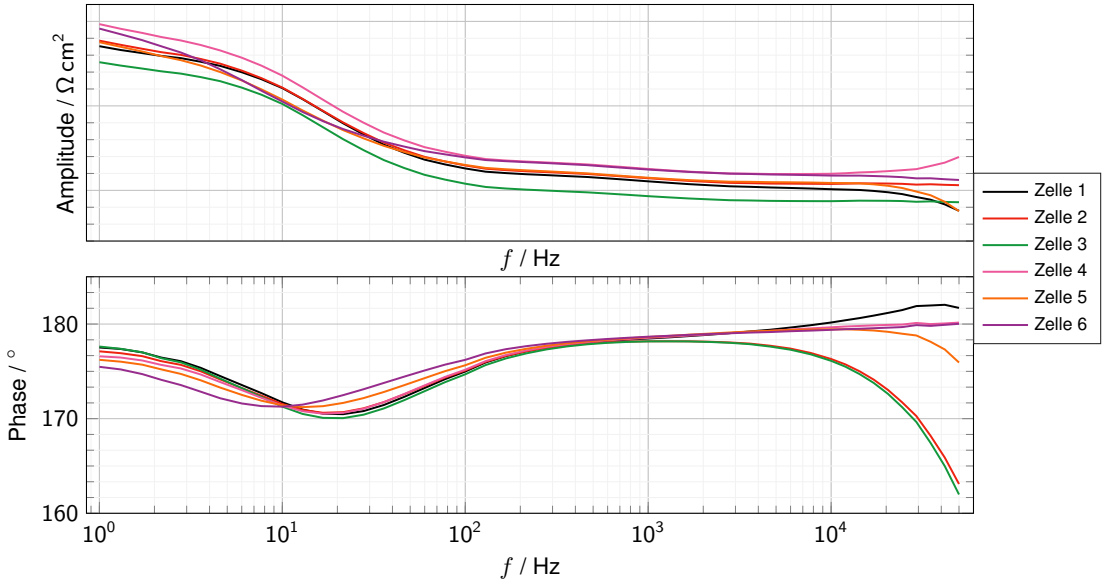

Abbildung 9.18: Amplituden- und Phasengang der finalen Messung nach der erneuten Konditionierung.

Auch die Ortskurven lassen sich wieder daraus ableiten, wie Abbildung 9.19 verdeutlicht.

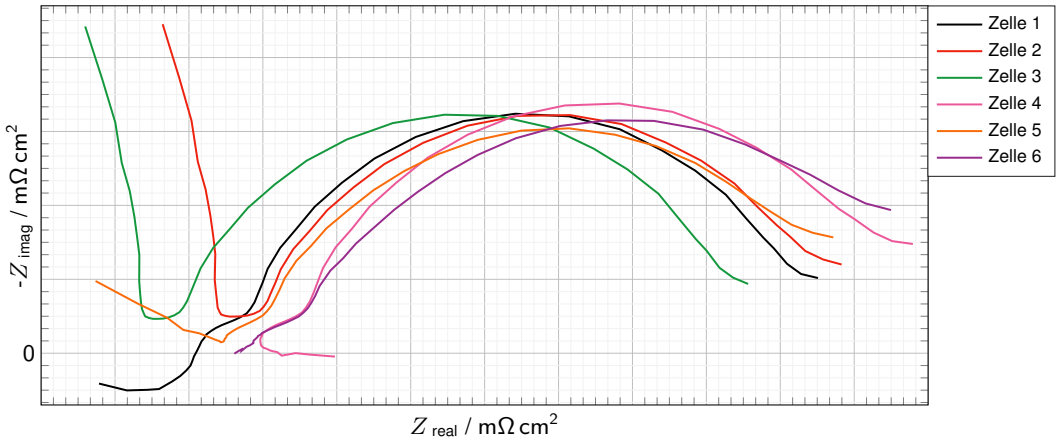

Abbildung 9.19: Ortskurven der finalen Messung nach der erneuten Konditionierung.

Das kapazitive Verhalten der Zellen 3 und 4 ist in der Zunahme des imaginären Anteils am linken Bildrand klar ersichtlich. Auch der zunächst kapazitive und dann induktive Verlauf der Zelle 1 lässt sich in der

dreimaligen Änderung der Kurvenkrümmung deutlich von den anderen Zellen unterscheiden. Je größer der Phasendrift, desto großer ist auch die Zunahme des imaginären Widerstandes. Ein Phasendrift wirkt sich stärker auf den realen und damit ohmschen Widerstand aus. Dies lässt sich anhand von Zelle 4 und 6 nachvollziehen. Deren Phasengang ist im Bereich mit hohen Frequenzen annähernd ident. Im Amplitudenplot unterscheiden sich diese, da Zelle 4 am Ende des Frequenzbandes eine Zunahme der Impedanz zeigt, wie dies in Abbildung 9.18 nachvollzogen werden kann.

Hierbei zeigt sich die Zunahme der Impedanz von Zelle 4 auch in einer Zunahme des Widerstandes auf der realen Achse. Die Abnahme, welche Zelle 6 zeigt, führt auch zu einer Verschiebung in Richtung geringerem Realteil. Insgesamt zeigt sich eine deutliche Homogenisierung der Einzelzellen, besonders wenn man diese mit der ersten Messung in Abbildung 9.11 vergleicht. Während Zelle 3 sich über die gesamte Prüfungsdauer nur geringfügig verändert, zeigen Zelle 4 und 6 eine Verbesserung um einen Faktor im zweistelligen Bereich. Es zeigt sich somit, dass die Differenz zwischen den Zellen durch eine längere Konditionierungsphase signifikant verbessert werden kann, zumindest bei diesem Prototyp.

10 ZUSAMMENFASSUNG UND AUSBLICK

Die Motivation hinter dieser Arbeit ist es, Impedanzmessung auch auf Stacks im industriellen Maßstab anwenden zu können. Durch den Eigenbau der Messeinheit erhält man den vollen Zugriff auf die Rohdaten der Messung, weshalb auch die Auswertung dieser Daten selbst durchgeführt werden muss. Wie die Ergebnisse zeigen, ist eine Auswertung der Messdaten gelungen, wobei der Algorithmus immer wieder angepasst werden musste, was den fortlaufenden Verbesserungen und Anpassungen an der Hardware und dem Messablauf geschuldet ist. Somit können von der HyCentA Research GmbH fortan Impedanzmessungen in einer Leistungsklasse angeboten werden, die so kommerziell nicht erhältlich sind.

Die Arbeit lässt sich grob in vier große Abschnitte unterteilen. Die Datenauswertung, die Kalibrierung der Sensoren, die Datenbereitstellung und die Datenanalyse.

10.1 Zusammenfassung

Im Abschnitt der Datenauswertung werden zuerst die beiden Darstellungsvarianten von komplexen Zahlen erläutert, anhand derer die wichtigsten Graphen der Impedanzspektroskopie erstellbar sind. Auch deren geometrischer Zusammenhang und wie diese ineinander überführt werden können, wird dargelegt. Die Abtastrate spielt in zweierlei Hinsicht eine wichtige Rolle. Sie bestimmt über das Nyquist-Shannon-Abtasttheorem bis zu welcher Frequenz ein Signal zuverlässig aufgelöst und damit wiederherstellbar ist. Außerdem wird, bei konstant gehaltener Messdauer, darüber die Anzahl der Datenpunkte einer Messung beeinflusst und damit welche Auflösung der Frequenzvektor in der Frequenzdomäne besitzt. Die Überführung der Zeitdomänedaten erfolgt mittels Fourier-Transformation, welche für jeden Punkt im Frequenzspektrum eine komplexe Zahl zurückgibt. Aus den Absolutwerten dieser Zahlen, ergibt sich ein Amplitudenvektor, der das Spektrum gespiegelt um die Mitte und mit halber Amplitudenhöhe ausgibt. Durch eine Spiegelung dieses Vektors erhält man den korrekten Amplitudenausschlag. Mit der Position des höchsten Amplitudenwertes in diesem Vektor kann man im Frequenzvektor an der selben Position die zum Signal zugehörige Frequenz ermitteln. Dieser Frequenzvektor ergibt sich aus der Abtastrate als höchste Frequenz und wird mit der Anzahl der Messpunkte diskretisiert. Auch hier ist eine Spiegelung notwendig, damit dieser dem Nyquist-Shannon-Theorem entspricht. Der Phasenwinkel kann über den trigonometrischen Zusammenhang aus der komplexen Zahl an der ermittelten Position im Ergebnisvektor berechnet werden. Somit ist es möglich das gemessene Signal zu charakterisieren. Da es sich um ein physische Messobjekt handelt, durchläuft der Stack eine Einschwingphase und die Messung enthält ein Signalrauschen. Damit der Einfluss dieser Faktoren so gering wie möglich gehalten wird, kommen Fensterfunktionen zum Einsatz. Diese filtern solche Phänomene durch Überlagerung des Messsignales mit einer Kurvenfunktion heraus, wobei sich für die Genauigkeit der Amplitude das Flattop als bestes herauskristallisiert hat. Dies wurde durch nummerische Versuche sowie Literatur unterlegt. Ein weiteres Phänomen im Zusammenhang mit der Fourieranalyse ist die spektrale Leckage. Eine Ursache für diese Fehler ist die Verarbeitung von Signalen mit unvollständigen Perioden, die sich aufgrund der Messdauer ergeben. Dadurch wird die Amplitudenhöhe beeinflusst und nicht korrekt ausgegeben. Durch den Einsatz von Fensterfunktionen kann diesem Phänomen vorgebeugt werden, da Beginn sowie das Ende des Datensatzes nicht verarbeitet werden. Ein weiterer Grund ist die Auflösung des Frequenzvektors beziehungsweise wenn die Signalfrequenz nicht einem Wert

in diesem Vektor entspricht. Hier wurde gezeigt, dass die Abweichung des Amplitudenwertes proportional zur Entfernung der Signalfrequenz zum nächsten Frequenzwert im Vektor ist. Für die Kompensation des Frequenzganges einiger Komponenten im Messaufbau werden noch einige Verfahren, wie beispielsweise die lineare Interpolation sowie die Polynom und Spline Regression, zum Annähern von Daten erläutert, um aus den diskreten Messpunkten eine kontinuierliche Funktion zu erhalten.

Wie bereits erwähnt, besitzen einige der verbauten Komponenten im Messaufbau ein frequenzabhängiges Verhalten und bringen so einen frequenzabhängigen Messfehler ins System ein. Um den Fehler dieser Komponenten kompensieren zu können, muss deren Frequenzverhalten bestimmt werden. Dies wird mittels Signalgenerator und einem Messshunt als Referenzgröße erreicht. Über die Differenz zwischen dem Shunt und den verbauten Bauteilen in der Messkette kann der Phasengang bestimmt werden. Da auch der Hall-Stromsensor ein solches Verhalten zeigt und der Signalgenerator nur kleine Ströme zur Verfügung stellen kann, wird die stromführende Leitung zwölfmal durch diesen hindurch und anschließend zum Shunt geführt. Dadurch wird mit geringem Aufwand ein höherer Messwert erzeugt und das Frequenzverhalten des Bauteiles kann gemessen werden. Somit ist es möglich, die Fehler zu kompensieren, indem man diese bei der Analyse der Messdaten um den Fehler bereinigt.

Für die Darstellung der Daten gibt es zwei etablierte Darstellungsformen. Zum einen das Auftragen der Amplituden sowie des Phasenwinkels über die eingeprägten Frequenzen, welche als Bode Plot bezeichnet wird. Hierbei sind alle Informationen über die Signale klar ersichtlich, wobei sich der Offset und die Höhe der Amplitude überlagern. Zum anderen über das Auftragen des Real- und Imaginärteils auf symmetrisch skalierten Achsen, wodurch sich die charakteristischen Ortskurven, die als Nyquist Plot bezeichnet werden, ergeben. Diese Darstellungsform beinhaltet keine Frequenzwerte, weshalb die dekadischen Werte nach Bedarf eingeblendet werden können. Außerdem erfolgt die Berechnung des Hochfrequenzwiderstandes innerhalb der Ortskurve. Der Nulldurchgang wird über die Geradengleichung zwischen dem ersten Punkt unter- sowie oberhalb der Nulllinie ermittelt. Durch die Frequenzen dieser beiden Punkte, sowie deren Abstand zur Nulllinie, wird linear interpoliert, wodurch sich die Frequenz des Nulldurchganges abschätzen lässt. Außerdem wird durch Bereitstellung von interaktiven Grafiken ermöglicht, dass die exakten Messwerte sowie die Frequenz für jeden Messpunkt dargestellt werden können.

Für die Datenanalyse stehen gemessene sowie berechnete Werte zur Verfügung. Anhand dieser können Ersatzschaltbilder entwickelt und angenähert werden, wodurch sich die Kurve durch die Komponentenwerte beschreiben lässt und auch Veränderungen einem numerischen Wert zugeordnet und damit verglichen werden können. Zum Verständnis des Einflusses einer Änderung der Komponentenwerte wird ein einfaches Ersatzschaltbild bestehend aus einem Membranwiderstand sowie zwei RC-Gliedern simuliert. Durch gezielte Veränderung einzelner Werte kann deren Einfluss auf die Kurvenform dargestellt werden. Diese dienen als Interpretationshilfe und zur Rückführung von Änderungen in der Kurvenform auf Komponenten des Membran Elektroden Aufbaues. Auch der Verlauf des Hochfrequenzwiderstandes über die Einzelzellen ist darstellbar. Durch die Archivierung der Analysedaten in einem offenen Dateiformat können die Ergebnisse auch mit anderen Programmen und zu einem späteren Zeitpunkt wiederverwendet beziehungsweise weitere Auswertungen vorgenommen werden.

10.2 Reflexion und Ausblick

Die in der Arbeit beschriebenen Auswerte- und Darstellungsalgorithmen stellen nur den derzeitigen Stand dar. Wie die ersten Messungen gezeigt haben, ergeben sich während dem Testbetrieb immer wieder Einflüsse die so nicht vorhersehbar sind. Beispiel hierfür ist etwa das Auffüllen des Wasserreservoirs, was zu einer Änderung der Leitfähigkeit im System führt. Erfolgt dies während einer Impedanzmessung, wird dies

auch in den Daten durch einen Drift der gesamten Messung sichtbar. Die direkte Folge daraus ist, dass die FFT eine falsche Frequenz ausgibt und bei der Beschneidung der Daten auf volle Periodenzahl der komplette Datensatz verworfen wurde und zu einem Abbruch des Programms geführt hat. Dies zeigt, dass für die Auswertung auch Einflüsse aus dem Teststand berücksichtigt werden müssen. Deshalb dient der jetzige Stand als Basis, anhand derer die spezifischen Anpassungen für jeden Stack gemacht werden.

Was im Umfang dieser Arbeit nicht gelungen ist, ist unterschiedliche Stacks bei verschiedenen Betriebspunkten miteinander zu vergleichen. Dies ist vor allem an den Stacks selbst festzumachen. Die äußerst schnell fortschreitende Degradation des Recycalyse-Stack lässt keine aussagekräftigen Vergleiche zu, da die Änderung nicht eindeutig zugeordnet werden kann. Am HTEC-Stack gibt es die Problematik mit der schlechten Zugänglichkeit und der damit verbunden Kontaktierungsprobleme. Auch der Kundenstack hat aufgrund seiner stetigen Veränderung über die Testdauer keine eindeutige Aussage über den Einfluss der Betriebsparameter zugelassen.

Aufgrund der schlechten Stacks konnte leider keine Versuchsreihe durchgeführt werden, anhand derer man verschiedene Betriebszustände und deren Auswirkung auf die Ortskurve untersucht. Nur die Messungen am HTEC-Stack zeigen, dass eine Erhöhung der Betriebsmitteltemperatur zu einer Verschiebung der Kurve auf der realen Achse führt. Mit einem Stack der reproduzierbare Messungen zulässt, könnten die Einflüsse von Kathoden- und Anodendruck sowie des Massenstromes untersucht und klassifiziert werden.

Die Interpretation der Impedanzspektroskopie an Elektrolysestacks bietet ein breites Forschungsfeld, da auch die Literatur zu dieser Thematik nur spärlich vorhanden ist. Deshalb wird am HyCentA an dieser Technologie weitergeforscht, um den Erkenntnisgewinn aus den Daten zu festigen und zu vertiefen. Schon jetzt ist diese Art der Messung prädestiniert für den kommerziellen Einsatz, da es möglich ist, die Zellen während des Betriebs zu überwachen. Mit dem zu erwartenden Wachstum in der Branche wird sich ein breites Einsatzgebiet ergeben, weshalb die Grundlagenforschung jetzt mehr denn je im Mittelpunkt steht.

LITERATURVERZEICHNIS

Gedruckte Werke

Achim Zielesny (2016). *From Curve Fitting to Machine Learning: An Illustrative Guide to Scientific Data Analysis and Computational Intelligence*. 2. Aufl. Springer (siehe S. 31).

Agate Martin, Patrick Trinke, Boris Bensmann, Richard Hanke-Rauschenbach (2022). „Hydrogen Crossover in PEM Water Electrolysis at Current Densities up to 10 A cm-2“. In: *Journal of The Electrochemical Society* 169.9 (siehe S. 12).

Alexander Trattner, Manfred Klell, Fabian Radner; (2022). „Sustainable hydrogen society - Vision, findings and development of a hydrogen economy using the example of Austria“. In: *International Journal of Hydrogen Energy* 47 (siehe S. 4).

Alfredo Ursua, Luis M. Gandia, Pablo Sanchis; (2012). „Hydrogen Production form Water electrolysis: Current Status and future Trens“. In: *Proceedings of the IEEE, Volume 100, Issue: 2* (siehe S. 8).

Andrzej Lasia (2014). *Electrochemical Impedance Spectroscopy and its Applications*. 1. Aufl. Springer Vieweg (siehe S. 36).

August Wilhelm von Hofmann (1866). *Introduction to Modern Chemistry: Experimental and Theoretic*. 1. Aufl. Walton und Marberly (siehe S. 8).

C. Immerz, B. Bensmann, P. Trinke, M. Suermann and R. Hanke-Rauschenbach (2018). „Local Current Density and Electrochemical Impedance Measurements within 50 cm Single-Channel PEM Electrolysis Cell“. In: *Journal of the Electrochemica Society* 165 (siehe S. 49, 76).

C. Rozain, P. Millet (2014). „Electrochemical characterization of Polymer Electrolyte Membrane Water Electrolysis Cells“. In: *Electrochimica Acta* 131 (siehe S. 49).

Clemens Heitzinger (2022). *Algorithms with JULIA; Optimization, Machine Learning, and Differential Equations Using the JULIA Language*. Springer (siehe S. 61).

D. Guilbert and G. Vitale (2019). „Dynamic Emulation of a PEM Electrolyzer by Time Constant Based Exponential Model“. In: *International Conference on Environment and Electrical Engineering* 18 (siehe S. 49).

Eleanor Chu, Alan George (2000). *Inside The FFT Black Box: Serial And Parallel Fast Fourier Transform Algorithms*. Computational Mathematics Series. CRC Press (siehe S. 15).

F.N. Khatib, Tabbi Wilberforce, Oluwatosin Ijaodola, Emmanuel Ogungbemi, Zaki El-Hassan, A. Durrant, J. Thompson, A.G. Olabi; (2019). „Material degradation of components in polymer electrolyte membrane (PEM) electrolytic cell and mitigation mechanisms: A review“. In: *Renewable and Sustainable Energy Reviews* 111 (siehe S. 36).

Garry McCracken, Peter Stott (2013). *Fusion, The Energy of the Universe*. 2. Aufl. Elsevier (siehe S. 7).

Guilbert, Damien and Vitale, Gianpaolo (2020). „Variable Parameters Model of a PEM Electrolyzer Based Model Reference Adaptive System Approach“. In: *IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe* (siehe S. 50).

Haas Christoph; (2018). „Entwicklung eines 3D-CFD Simulationsmodells eines PEM-Elektrolysemoduls“. Masterarbeit. TU Graz (siehe S. 12).

HyCentA Research GmbH (2022). „Electrochemical Characterisation of Electrolysis Stacks“. In: (Siehe S. 6).

Jean-Baptiste Jorcin and Mark E. Orazem and Nadine Pébère and Bernard Tribollet (2006). „CPE analysis by local electrochemical impedance spectroscopy“. In: *Electrochimica Acta* 51 (siehe S. 45).

Johannes Töpler, Jochen Lehmann (eds.) (2017). *Wasserstoff und Brennstoffzelle: Technologien und Marktperspektiven*. 2. Aufl. Springer Vieweg (siehe S. 7).

Joshua Eder (2023). „Analyse und Optimierung von Elektrolysestacks“. Masterarbeit. FH Campus02 (siehe S. 6, 54, 55).

K. Elsøe, L. Grahl-Madsen, G. G. Scherera, J. Hjelm, M. B. Mogensen; (2017). „Electrochemical Characterization of PEMEC Using Impedance Spectroscopy“. In: *The Electrochemical Society* (siehe S. 36).

Lothar Papula (2011). *Mathmatik für Ingeniere und Naturwissenschaftler Band 1*. 13. Aufl. Springer Vieweg (siehe S. 16).

M. Durand, A. Picot, J. Rénier, C. Turbin, O. Crassous, M. Scohy, R. Stephan, O. Abassie and C. Andrieux (2021). „Automated Analysis of EIS curves for PEM Fuel Cells using Dynamic Time Warping“. In: *International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives* 13 (siehe S. 41).

Manfred Klell, Helmut Eichlseder, Alexander Trattner; (2018). „Wasserstoff in der Fahrzeugtechnik“. In: *ATZ/MTZ-Fachbuch* (siehe S. 7, 9–11).

Martin Meyer (2017). *Signalverarbeitung - Analoge und digitale Signale, Systeme und Filter*. 8. Aufl. Springer Vieweg (siehe S. 15, 25, 26).

Michael Cerna, Audrey F. Harvey; (2000). „The Fundamentals of FFT-Based Signal Analysis and Measurement“. In: *Application Note 041* (siehe S. 25, 26).

Michael Möser (2018). *Digitale Signalverarbeitung in der Messtechnik*. 1. Aufl. Springer Vieweg (siehe S. 23).

Michael Sterner, Ingo Stadler (2017). *Energiespeicher Bedarf, Technologien, Integration*. 2. Aufl. Springer Vieweg (siehe S. 2, 3).

Ömer Faruk Selamet, Fatih Becerikli, Mahmut D. Mat, Yüksel Kaplan (2011). „Development and testing of a highly efficient proton exchange membrane (PEM) electrolyzer stack“. In: *International Journal of Hydrogen Energy* 36 (siehe S. 80).

Patrick Schnabel (2014). *Elektronik-Fibel: Grundlagen, Bauelemente, Schaltungstechnik, Digitaltechnik*. Elektronik Kompendium (siehe S. 37).

Peter Kurzweil (2020). *Angewandte Elektrochemie: Grundlagen, Messtechnik, Elektroanalytik, Energiewandlung, technischer Verfahren*. Springer Vieweg (siehe S. 38).

Peter Kurzweil, Otto K. Dietlmeier (auth.) (2015). *Elektrochemische Speicher: Superkondensatoren, Batterien, Elektrolyse-Wasserstoff, Rechtliche Grundlagen*. Springer Vieweg (siehe S. 10).

Reinhard Lerch (2007). *Elektrische Messtechnik*. 4. Aufl. Springer Vieweg (siehe S. 16).

Rohde & Schwarz (2017). „Der normgerechte Umgang mit Größen, Einheiten und Gleichungen“. In: (Siehe S. 67).

S. Siracusano, A. Di Blasi, V. Baglio, G. Brunaccini, N. Briguglio, A. Stassi, R. Ornelas, E. Trifoni, V. Antonucci, A.S. Arico; (2011). „Optimization of components and assembling in a PEM electrolyzer stack“. In: *International Journal of Hydrogen Energy* (siehe S. 36).

Shiva Kumar, S. und V. Himabindu; (2019). „Hydrogen production by PEM water electrolysis – A review“. In: *Materials Science for Energy Technologies* 2 (siehe S. 10).

Shucheng Sun, Zhigang Shao, Hongmei Yu, Guangfu Li, Baolian Yi; (2014). „Investigations on degradation of the long-term proton exchange membrane water electrolysis stack“. In: *Journal of Power Sources* (siehe S. 36).

Steven L. Brunton, J. Nathan Kutz (2017). *Data Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press (siehe S. 18–22, 33).

Suermann, Michel and Beard Patru, Alexandra and Schmidt, Thomas and Büchi, Felix (2017). „High pressure polymer electrolyte water electrolysis: Test bench development and electrochemical analysis“. In: *International Journal of Hydrogen Energy* (siehe S. 75).

Tom Smolinka, Martin Günther, Jürgen Garche; (2011). „Stand und Entwicklungspotenzial der Wasserelektrolyse zur Herstellung von Wasserstoff aus regenerativen Energien“. In: *Kurzfassung des Abschlussberichts* (siehe S. 4, 11).

Ulrich Karrenberg (2016). *Signale Prozesse Systeme*. 7. Aufl. Springer Vieweg (siehe S. 18).

V. Ragone (1968). „Review of Battery Systems for Electrically Powered Vehicles“. In: *SAE paper 680453* 1 (siehe S. 2).

Wolfgang Demtröder; (2016). „Experimentalphysik 3: Atome, Moleküle und Festkörper“. In: *Springer-Lehrbuch* (siehe S. 7).

Online-Quellen

Europäische Kommision, CORDIS Forschungsergebnisse der EU (2023). *New sustainable and recyclable catalytic materials for proton exchange membrane electrolysers*. URL: `https://cordis.europa.eu/project/id/861960/de`. (accessed: 29.03.2023) (siehe S. 6).

FFTs Paket Dokumentation (2023). *AbstractFFTs.jl*. URL: `https://juliamath.github.io/AbstractFFTs.jl/stable/api/`. (accessed: 05.01.2023) (siehe S. 69).

GeoSphere Austria – Bundesanstalt für Geologie, Geophysik, Klimatologie und Meteorologie (2022). *AbstractFFTs.jl*. URL: https://www.zamg.ac.at/cms/de/topmenu/impressum. (accessed: 29.03.2023) (siehe S. 1).

GitLab B.V (2023). *The DevSecOps Platform*. URL: https://about.gitlab.com/platform/?stage=plan. (accessed: 29.03.2023) (siehe S. 6).

Izaak Neutlings (2021). *Complex plane and oscillator*. URL: https://tikz.net/complex/. (accessed: 08.08.2022) (siehe S. 16).

Izaak Neutlings (2021). *Fourier series and synthesis*. URL: https://tikz.net/fourier_series/. (accessed: 08.08.2022) (siehe S. 18, 19).

J. Nathan Kutz (2015). *Data Fitting: Polynomial Fitting and Splines, Part 1*. University of Washington. URL: https://www.youtube.com/watch?v=BqZXS3n75l0. (accessed: 1.11.2022) (siehe S. 34).

J. Nathan Kutz (2015). *Data Fitting: Polynomial Fitting and Splines, Part 4*. University of Washington. URL: https://www.youtube.com/watch?v=rtw0rZL02M0. (accessed: 1.11.2022) (siehe S. 35).

Julia Lab at MIT (2023). *The Julia Programming Language*. URL: https://julialang.org/. (accessed: 29.03.2023) (siehe S. 6).

Matt Lacey (2022a). *Simple circuits with resistors and capacitors*. URL: http://lacey.se/science/eis/simple-circuits/. (accessed: 09.12.2022) (siehe S. 53).

– (2022b). *The constant phase element*. URL: http://lacey.se/science/eis/constant-phase-element/. (accessed: 28.11.2022) (siehe S. 45).

– (2023). *Diffusion impedance*. URL: http://lacey.se/science/eis/diffusion-impedance/. (accessed: 30.03.2023) (siehe S. 46).

PalmSens Knowledge Base (2022). *Bode Plot*. PalmSens. URL: https://www.palmsens.com/knowledgebase-topic/bode-plot/#:~:text=The%20Bode%20Plot%20is%20one,plotted%20versus%20the%20lg%20f.. (accessed: 13.11.2022) (siehe S. 39).

Research Solutions & Resources LLC (2022). *Explaining a Constant Phase Element (CPE)*. URL: http://consultrsr.net/resources/eis/cpe2.htm#ref. (accessed: 28.11.2022) (siehe S. 45).

Statista (2022). *Verteilung der CO2-Emissionen weltweit nach Sektor bis 2020*. URL: https://de.statista.com/statistik/daten/studie/167957/umfrage/verteilung-der-co-emissionen-weltweit-nach-bereich/#professional. (accessed: 31.07.2022) (siehe S. 1).

Statistik Austria (2022). *Bestand 2021 nach Fahrzeugart*. URL: https://www.statistik.at/statistiken/tourismus-und-verkehr/fahrzeuge/kfz-bestand. (accessed: 01.08.2022) (siehe S. 2).

Umweltbundesamt (2022). *Erdüberlastungstag: Ressourcen für 2022 verbraucht*. URL: https://www.umweltbundesamt.de/themen/erdueberlastungstag-ressourcen-fuer-2022-verbraucht. (accessed: 31.07.2022) (siehe S. 1).

Umweltbundesamt (2021). *Treibhausgas-Bilanz 2019 nach Sektoren*. URL: https://www.umweltbundesamt.at/news210119/sektoren. (accessed: 01.08.2022) (siehe S. 1).

VCÖ (2020). *VCÖ Factsheet: Energiewende im Verkehr rascher voranbringen*. URL: https://www.vcoe.at/publikationen/vcoe-factsheets/detail/vcoe-2019-10-energiewende-im-verkehr-rascher-voranbringen#:~:text=Der%20Energiebedarf%20des%20Verkehrs%20in,viel%20wie%20im%20Jahr%201990.. (accessed: 01.08.2022) (siehe S. 2).

Voestalpine (2021). *Umwelt Energie*. URL: https://reports.voestalpine.com/2020/cr-bericht/umwelt/energie.html. (accessed: 01.08.2022) (siehe S. 2).

IG-Windkraft (2022). *Windenergie in Österreich*. URL: https://www.igwindkraft.at/fakten/?xmlval_ID_KEY%5B0%5D=1234. (accessed: 01.08.2022) (siehe S. 2).

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

EIS	Elektrochemische Impedanzspektroskopie
PEM	Proton exchange membrane
CPE	Constant phase element

ZUSÄTZLICHE ERGEBNISSE

A.1 Zweite Messung mit 60 % höherem Druck

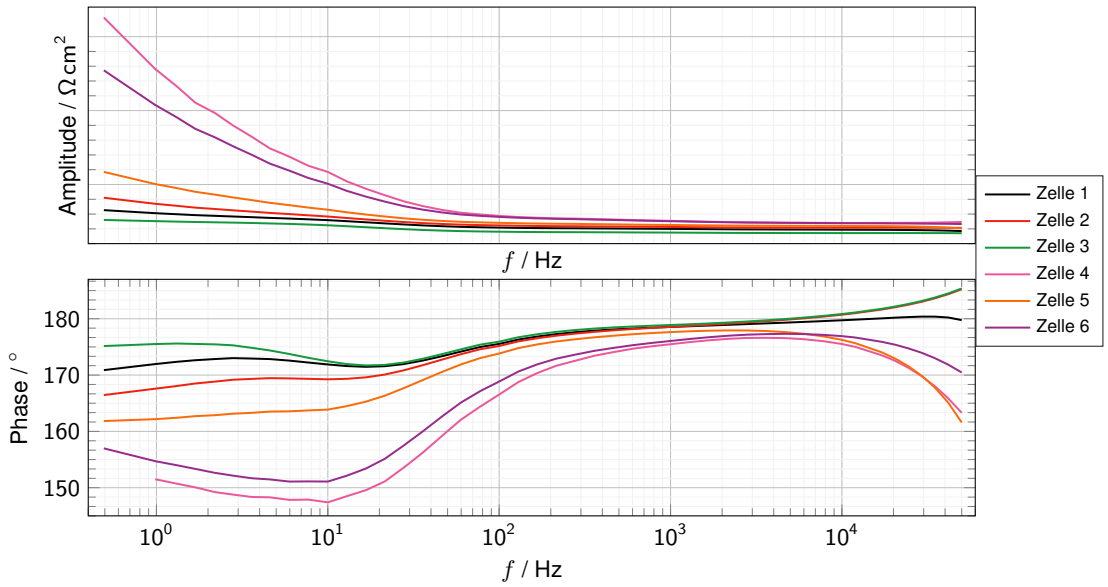

Abbildung A.1: Amplituden- und Phasengang der zweiten Messung mit 60 % höherem Druck im Vergleich zur Eingangsmessung.

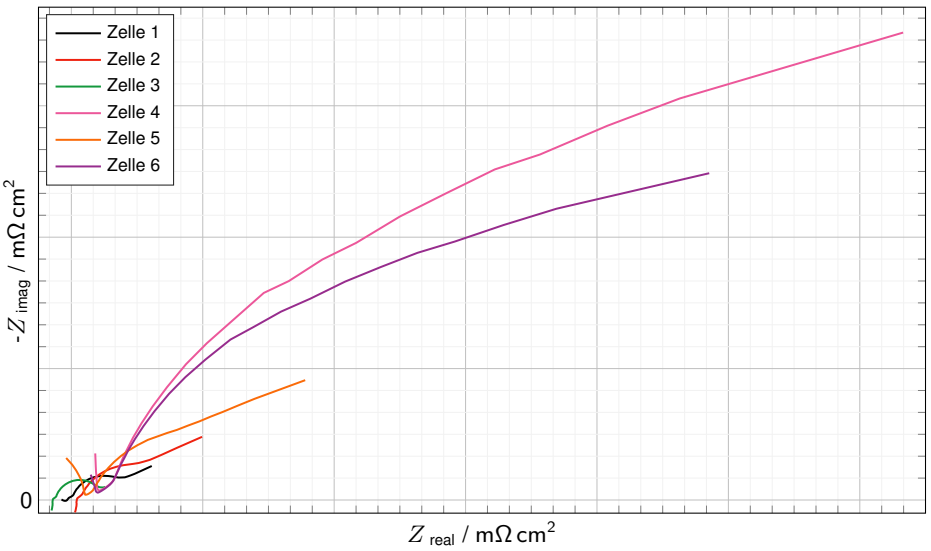

Abbildung A.2: Ortskurve der zweiten Messung mit 60 % höherem Druck im Vergleich zur Eingangsmessung.

HAUPTPROGRAMM STACK EVALUATION

```
# --- ClemensHyCentaStackEvaluation.jl ---------------------------------------------------------------------------------------------
# Collection of functions to analyse harmonic signals in time series data.
#
# Copyright (C) 2022    Stefan N. Pofahl (HyCentA, Graz, Austria)
#
# File Owner:
#    - Clemens Weiskopf
#
# History:
# v1.0 (25-Okt-2022), Clemens Weiskopf
#    - Release of new Version
# v1.1 (02-Nov-2022), Clemens Weiskopf
#    - Implemented MyLibCheckPlotParam Function
# ------------------------------------------------------------------------------------------------------------------------------------
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
# ------------------------------------------------------------------------------------------------------------------------------------
# Markdown notation: https://docs.julialang.org/en/v1/stdlib/Markdown/
# ------------------------------------------------------------------------------------------------------------------------------------
#addpath_genpath(raw"Folder_Path")
using PlotlyJS
using ClemensHyCentaSignalAnalysis, ClemensHyCentaFunctionsForStackEvaluation, ClemensHyCentaPlotFunctions
using Printf, MAT, StatsBase, ProgressMeter, CubicSplines, JLD2, DSP, HDF5

###-----------------------------------------------------------------------------------------------------------------------------------
# Paths for Data and Results
#-----------------------------------------------------------------------------------------------------------------------------------
Raw_Data_Path            = raw"D:\Messdaten\20230227_No1_1_Activation"
Results_Folder_Path      = raw"D:\Messdaten\20230227_No1_1_Activation_Ergebnisse"
Calibration_Results_Path = raw"D:\Messdaten\Full-Spec_100A-DC_10A-AC_With_COIL\Calibrationdata"

# structure of hdf5 file
# h5open(raw"D:\Messdaten\20230227_No1_1_Activation\0021EIS_49p0A000035p89EXC_EIS_60C_EXC_20230227_144203.hdf5", "r")

###-----------------------------------------------------------------------------------------------------------------------------------
# Parameter for FFT
#-----------------------------------------------------------------------------------------------------------------------------------
FFT_window_func     = "flattop" # one of: "rect", "hamming", "hann", "kaiser77", "flattop", "tukey", "blackmanharris"
                                # Kaiserparameter can be varied within the String: "kaiser28", "kaiser55", ...
FFT_window_func     = MyLibCheckFFTInputParam(FFT_window_func)
FFT_Trim_of_Signal  = 1  # in Seconds, Time which get cut away from the beginning of the Measurementdata
Full_Spectrum       = true # true if full spectrum, false if short spectrum

###-----------------------------------------------------------------------------------------------------------------------------------
# Parameter for Plots
#-----------------------------------------------------------------------------------------------------------------------------------
Dataformat                      = ".html" # plot format, one of: ".html", ".pdf", ".svg"
Transparent_Background          = false
Transparent_Background_Legend   = false
Plot_Size                       = true # true for whole page and false for half page
Show_Grid                       = true
Show_Meta_Information           = true # show frequency information in nyquist plot
Calc_HFR                        = true # calculation of high frequency resistance
# Limit the Frequencies to certain Values in Nyquist Plot
Border_Freq = 15000

Plotparameter = Dict("Dataformat" => Dataformat, "Meta_Information" => Show_Meta_Information, "Transparent_Background" => Transparent_Background,
                     "Plot_Size" => Plot_Size, "Show_Grid" => Show_Grid, "Transparent_Background_Legend" => Transparent_Background_Legend,
                     "Full_Spectrum" => Full_Spectrum, "b_HFR_on" => Calc_HFR)
Plotparameter = MyLibCheckPlotParam(Plotparameter) # checking Plotparameter
###-----------------------------------------------------------------------------------------------------------------------------------
# Path for safing FFT Results
#-----------------------------------------------------------------------------------------------------------------------------------
Results_Path, Plots_Path, Plots_Path_AmplitudeAngle, Plots_Path_Nyquist,
Plots_Path_Current      = MyLibCheckandmakeDirectories(FFT_window_func, Raw_Data_Path, Results_Folder_Path)
FFT_Results_Datasafe    = joinpath(Results_Path , String("FFT_Results_" * FFT_window_func * ".mat"))
```

```
#h5open(Raw_Data_Path_Files[1], "r")
# load only Measurement File, no Noise Files
Raw_Data_Path_Files, Active_Cell_Area, Cells_per_Segment, Check_Sampling_Frequency = MyLibSearchforMeasurementFiles(Raw_Data_Path)
@info("Active_Cell_Area: " * string(Active_Cell_Area) * " and " * "Cells_per_Segment: " * string(Cells_per_Segment))

###-----------------------------------------------------------------------------------------------------------------------
# Calibration Parameter VeriVolt
#-----------------------------------------------------------------------------------------------------------------------
Calibration_Spline_Amplitude_Path   = joinpath(Calibration_Results_Path ,String("Calibration_Parameter_VeriVolt_Amplitude_" * FFT_window_func * ".jld2"))
Calibration_Spline_Amplitude        = load(Calibration_Spline_Amplitude_Path, "Amplitude_Fit")
Spline_Border_Frequency_Amplitude   = load(Calibration_Spline_Amplitude_Path, "Border_Frequency")

Calibration_Spline_Phase_Path       = joinpath(Calibration_Results_Path ,String("Calibration_Parameter_VeriVolt_Phase_" * FFT_window_func * ".jld2"))
Calibration_Spline_Phase            = load(Calibration_Spline_Phase_Path, "Phase_Fit")
Spline_Border_Frequency_Phase       = load(Calibration_Spline_Phase_Path, "Border_Frequency")

# check if calibration frequency is higher then signal frequency
if Spline_Border_Frequency_Amplitude < Check_Sampling_Frequency
    throw("EIS Frequency is higher then Calibration Frequency. No Amplitude Correction done for VeriVolt!")
elseif Spline_Border_Frequency_Phase < Check_Sampling_Frequency
    throw("EIS Frequency is higher then Calibration Frequency. No Phase Correction done for VeriVolt!")
end

# Embossing Frequency (Einpraegefrequenz) from Filename with lower and upper limit
Upper_Border_Embossing_Frequency = 1.05 # higher Limit of _Embossing_Frequency
Lower_Border_Embossing_Frequency = 0.95 # lower Limit of _Embossing_Frequency
###-----------------------------------------------------------------------------------------------------------------------
# FFT Analysis
#-----------------------------------------------------------------------------------------------------------------------
if ~isfile(FFT_Results_Datasafe)
    FFT_Stack_U_Ampl                = Vector{Float64}(undef, 0)
    FFT_Stack_U_Angle               = Vector{Float64}(undef, 0)
    FFT_Stack_U_Freq                = Vector{Float64}(undef, 0)

    FFT_Stack_I_Sum_Ampl            = Vector{Float64}(undef, 0)
    FFT_Stack_I_Sum_Angle           = Vector{Float64}(undef, 0)
    FFT_Stack_I_Sum_Freq            = Vector{Float64}(undef, 0)

    FFT_Cells_U_Ampl_temp           = Vector{Float64}(undef, 0)
    FFT_Cells_U_Angle_temp          = Vector{Float64}(undef, 0)
    FFT_Cells_U_Freq_temp           = Vector{Float64}(undef, 0)
    FFT_Cells_U_Ampl                = Matrix{Float64}(undef, 0, 0)
    FFT_Cells_U_Angle               = Matrix{Float64}(undef, 0, 0)
    FFT_Cells_U_Freq                = Matrix{Float64}(undef, 0, 0)

    Active_Cell_Area                = Float64
    Cells_per_Segment               = Float64
    Amplitude_Resistance_Cells      = Matrix{Float64}(undef, 0, 0)
    Amplitude_Resistance_Stack      = Vector{Float64}(undef, 0)
    Phase_Resistance_Cells          = Matrix{Float64}(undef, 0, 0)
    Phase_Resistance_Stack          = Vector{Float64}(undef, 0)
    Impedance_Cells_Real            = Matrix{Float64}(undef, 0, 0)
    Impedance_Cells_Imag            = Matrix{Float64}(undef, 0, 0)
    Impedance_Stack_Real            = Vector{Float64}(undef, 0)
    Impedance_Stack_Imag            = Vector{Float64}(undef, 0)

    Corrupt_Files                   = Vector{String}(undef, 0) # string for files that fail the signal analysis

    @showprogress 1 "FFT Data Analysis ..." for i_file in eachindex(Raw_Data_Path_Files)
        println("       Filename: ", Raw_Data_Path_Files[i_file])

        # Read Data from E-File
        global _Sampling_Freq, _Cells_U, _Stack_U, _Stack_I_Sum, Active_Cell_Area, Cells_per_Segment = MyLibReadHDF5(Raw_Data_Path_Files[i_file])

        # read embossing frequency from filename
        _Embossing_Frequency = MyLib_Read_Frequency_from_Filename(Raw_Data_Path_Files[i_file])

        # Centering Raw Data around mean for first Analysis
        _Stack_U_Estimation     = vec(_Stack_U     .- mean(_Stack_U))
        _Stack_I_Sum_Estimation = vec(_Stack_I_Sum .- mean(_Stack_I_Sum))

        # Determine Frequency of Raw Data
        _freq_Stack_U,     _, _amp_Stack_U, _, _, _     = MyLibFFT(_Stack_U_Estimation, _Sampling_Freq, 0, FFT_window_func, 0)
        _freq_Stack_I_Sum, _, _amp_Stack_I_Sum, _, _, _ = MyLibFFT(_Stack_I_Sum_Estimation, _Sampling_Freq, 0, FFT_window_func, 0)

        # Find Peakposition
        _index_ampl_Stack_U     = argmax(vec(_amp_Stack_U))
        _index_ampl_Stack_I_Sum = argmax(vec(_amp_Stack_I_Sum))

        # checking if frequency for signal trimming is not to far of form embossing frequency
        if _freq_Stack_I_Sum[_index_ampl_Stack_I_Sum] > _Embossing_Frequency * Lower_Border_Embossing_Frequency &&
            _freq_Stack_I_Sum[_index_ampl_Stack_I_Sum] < _Embossing_Frequency * Upper_Border_Embossing_Frequency &&
            _freq_Stack_U[_index_ampl_Stack_U] < _Embossing_Frequency * Upper_Border_Embossing_Frequency

            _Trimmed_Stack_U    = MyLibInputSignalTrim(_Sampling_Freq, _freq_Stack_U[_index_ampl_Stack_U], _Stack_U, FFT_Trim_of_Signal)
            _Trimmed_Stack_I_Sum = MyLibInputSignalTrim(_Sampling_Freq, _freq_Stack_I_Sum[_index_ampl_Stack_I_Sum], _Stack_I_Sum, FFT_Trim_of_Signal)

            # FFT for Signal Analysis with whole Periods
            _freq_Stack_U, _, _amp_Stack_U, _angle_Stack_U, _, _offset_Stack_U
                = MyLibFFT(_Trimmed_Stack_U, _Sampling_Freq, 0, FFT_window_func, 0)

            _freq_Stack_I_Sum, _, _amp_Stack_I_Sum, _angle_Stack_I_Sum, _, _offset_Stack_I_Sum
```

```
                = MyLibFFT(_Trimmed_Stack_I_Sum, _Sampling_Freq, 0, FFT_window_func, 0)

            # Find Peakposition
            _index_ampl_Stack_U          = argmax(vec(_amp_Stack_U))
            _index_ampl_Stack_I_Sum      = argmax(vec(_amp_Stack_I_Sum))

            # Evaluation of single Cells
            for cellnumber in eachindex(_Cells_U[1, :])

                # Centering Raw Data around mean for first Analysis
                _Cells_U_Estimation = vec(_Cells_U[:, cellnumber] .- mean(_Cells_U[:, cellnumber]))

                # Determine Frequency of Raw Data
                _freq_Cells_U, _, _amp_Cells_U, _, _, _ = MyLibFFT(_Cells_U_Estimation, _Sampling_Freq, 0, FFT_window_func, 0)
                _index_ampl_Cells_U = argmax(vec(_amp_Cells_U))

                # checkin if signal frequency is close to frequency from filename
                if _freq_Cells_U[_index_ampl_Cells_U] > _Embossing_Frequency * Lower_Border_Embossing_Frequency &&
                    _freq_Cells_U[_index_ampl_Cells_U] < _Embossing_Frequency * Upper_Border_Embossing_Frequency

                    # Trim Signal to whole Periods
                    _Trimmed_Cells_U = MyLibInputSignalTrim(_Sampling_Freq, _freq_Cells_U[_index_ampl_Cells_U], _Cells_U[:, cellnumber], FFT_Trim_of_Signal)

                    # FFT for Signal Analysis with whole Periods
                    _freq_Cells_U, _, _amp_Cells_U, _angle_Cells_U, _, _offset_Cells_U = MyLibFFT(_Trimmed_Cells_U, _Sampling_Freq, 0, FFT_window_func, 0)
                    _index_ampl_Cells_U = argmax(vec(_amp_Cells_U))

                    push!(FFT_Cells_U_Ampl_temp, _amp_Cells_U[_index_ampl_Cells_U])
                    push!(FFT_Cells_U_Angle_temp, _angle_Cells_U[_index_ampl_Cells_U])
                    push!(FFT_Cells_U_Freq_temp, _freq_Cells_U[_index_ampl_Cells_U])
                else
                    # save filename if file is corrupt
                    push!(Corrupt_Files, Raw_Data_Path_Files[i_file])
                end
            end
            if isempty(FFT_Cells_U_Ampl) && sum(occursin.(Raw_Data_Path_Files[i_file], Corrupt_Files)) == 0
                FFT_Cells_U_Ampl  = FFT_Cells_U_Ampl_temp
                FFT_Cells_U_Angle = FFT_Cells_U_Angle_temp
                FFT_Cells_U_Freq  = FFT_Cells_U_Freq_temp
            elseif sum(occursin.(Raw_Data_Path_Files[i_file], Corrupt_Files)) == 0
                FFT_Cells_U_Ampl  = hcat(FFT_Cells_U_Ampl, FFT_Cells_U_Ampl_temp)
                FFT_Cells_U_Angle = hcat(FFT_Cells_U_Angle, FFT_Cells_U_Angle_temp)
                FFT_Cells_U_Freq  = hcat(FFT_Cells_U_Freq, FFT_Cells_U_Freq_temp)
            end
            FFT_Cells_U_Ampl_temp  = []
            FFT_Cells_U_Angle_temp = []
            FFT_Cells_U_Freq_temp  = []

            # if cells contain corrupt measurements, they wont be saved
            if sum(occursin.(Raw_Data_Path_Files[i_file], Corrupt_Files)) == 0
                # Safe Results in Vector>
                push!(FFT_Stack_U_Ampl,          _amp_Stack_U[_index_ampl_Stack_U])
                push!(FFT_Stack_U_Angle,         _angle_Stack_U[_index_ampl_Stack_U])
                push!(FFT_Stack_U_Freq,          _freq_Stack_U[_index_ampl_Stack_U])

                push!(FFT_Stack_I_Sum_Ampl,      _amp_Stack_I_Sum[_index_ampl_Stack_I_Sum])
                push!(FFT_Stack_I_Sum_Angle,     _angle_Stack_I_Sum[_index_ampl_Stack_I_Sum])
                push!(FFT_Stack_I_Sum_Freq,      _freq_Stack_I_Sum[_index_ampl_Stack_I_Sum])
            end
        else
            # save filename if file is corrupt
            push!(Corrupt_Files, Raw_Data_Path_Files[i_file])
        end
    end

    ###-----------------------------------------------------------------------------------------------------------------
    # Correction of Measurementdata from VeriVolt Calibration
    #-----------------------------------------------------------------------------------------------------------------
    # Calculate the Amplitude calibration vector into the measurment
    Amplitude_Calibration_VeriVolt = Calibration_Spline_Amplitude[FFT_Stack_U_Freq]
    FFT_Stack_U_Ampl_calibrated = FFT_Stack_U_Ampl .* Amplitude_Calibration_VeriVolt

    #Calculate the Phase calibration vector into the measurment
    Phase_Calibration_VeriVolt = Calibration_Spline_Phase[FFT_Stack_U_Freq]
    #VeriVolt Voltage divider is lacking behind with the phase. That's why we have to add the calibration phase
    FFT_Stack_U_Angle_calibrated = FFT_Stack_U_Angle .- Phase_Calibration_VeriVolt
    #-----------------------------------------------------------------------------------------------------------------

    ###-----------------------------------------------------------------------------------------------------------------
    # Standardization of Stack Data from Cellarea
    #-----------------------------------------------------------------------------------------------------------------
    # Amplitude and Phase Resistance Stack
    Amplitude_Resistance_Stack  = FFT_Stack_U_Ampl_calibrated ./ FFT_Stack_I_Sum_Ampl .* (Active_Cell_Area / Cells_per_Segment)
    Phase_Resistance_Stack      = (FFT_Stack_U_Angle_calibrated .- FFT_Stack_I_Sum_Angle)
    unwrap!(Phase_Resistance_Stack, Phase_Resistance_Stack; range = pi)
    # Impedance Stack
    _Real_Part_Stack_temp = Amplitude_Resistance_Stack .* cos.(Phase_Resistance_Stack)
    _Imag_Part_Stack_temp = Amplitude_Resistance_Stack .* sin.(Phase_Resistance_Stack) .* im
    _Impedance_Stack      = _Real_Part_Stack_temp .+ _Imag_Part_Stack_temp
    # convert to degree
    Phase_Resistance_Stack = Phase_Resistance_Stack .* 180 / pi

    Impedance_Stack_Real = real.(_Impedance_Stack) .* -1
```

```
    Impedance_Stack_Imag = imag.(_Impedance_Stack)

    ###-----------------------------------------------------------------------------------------------------------
    # Komplex Values for Stack and
    #-----------------------------------------------------------------------------------------------------------
    for cellcounter in eachindex(FFT_Cells_U_Ampl[:, 1])
        Amplitude_Resistance_Cells_temp  = FFT_Cells_U_Ampl[cellcounter, :] ./ FFT_Stack_I_Sum_Ampl .* (Active_Cell_Area / Cells_per_Segment)
        Phase_Resistance_Cells_temp      = (FFT_Cells_U_Angle[cellcounter, :] .- FFT_Stack_I_Sum_Angle)
        unwrap!(Phase_Resistance_Cells_temp, Phase_Resistance_Cells_temp; range = pi)

        _Real_Part_Cells_temp            = Amplitude_Resistance_Cells_temp .* cos.(Phase_Resistance_Cells_temp)
        _Imag_Part_Cells_temp            = Amplitude_Resistance_Cells_temp .* sin.(Phase_Resistance_Cells_temp) .* im
        _Impedance_Cells                 = _Real_Part_Cells_temp .+ _Imag_Part_Cells_temp

        Impedance_Cells_Real_temp        = real.(_Impedance_Cells) .* -1
        Impedance_Cells_Imag_temp        = imag.(_Impedance_Cells)

        if isempty(Amplitude_Resistance_Cells)
            Amplitude_Resistance_Cells   = Amplitude_Resistance_Cells_temp
            Phase_Resistance_Cells       = Phase_Resistance_Cells_temp .* 180 / pi

            Impedance_Cells_Real         = Impedance_Cells_Real_temp
            Impedance_Cells_Imag         = Impedance_Cells_Imag_temp
        else
            Amplitude_Resistance_Cells   = hcat(Amplitude_Resistance_Cells, Amplitude_Resistance_Cells_temp)
            Phase_Resistance_Cells       = hcat(Phase_Resistance_Cells, Phase_Resistance_Cells_temp .* 180 / pi)

            Impedance_Cells_Real         = hcat(Impedance_Cells_Real, Impedance_Cells_Real_temp)
            Impedance_Cells_Imag         = hcat(Impedance_Cells_Imag, Impedance_Cells_Imag_temp)
        end
    end
    #-----------------------------------------------------------------------------------------------------------

    #----- Safe Data to File ---------------------------------------------------------------------------------
    # as .mat File
    matwrite(FFT_Results_Datasafe,
    Dict(
        "FFT_Stack_U_Ampl"              => FFT_Stack_U_Ampl_calibrated,
        "FFT_Stack_U_Angle"             => FFT_Stack_U_Angle_calibrated,
        "FFT_Stack_U_Freq"              => FFT_Stack_U_Freq,
        "FFT_Stack_I_Sum_Ampl"          => FFT_Stack_I_Sum_Ampl,
        "FFT_Stack_I_Sum_Angle"         => FFT_Stack_I_Sum_Angle,
        "FFT_Stack_I_Sum_Freq"          => FFT_Stack_I_Sum_Freq,
        "FFT_Cells_U_Ampl"              => FFT_Cells_U_Ampl,
        "FFT_Cells_U_Angle"             => FFT_Cells_U_Angle,
        "FFT_Cells_U_Freq"              => FFT_Cells_U_Freq,
        "Active_Cell_Area"              => Active_Cell_Area,
        "Cells_per_Segment"             => Cells_per_Segment,
        "Amplitude_Resistance_Stack"    => Amplitude_Resistance_Stack,
        "Phase_Resistance_Stack"        => Phase_Resistance_Stack,
        "Amplitude_Resistance_Cells"    => Amplitude_Resistance_Cells,
        "Phase_Resistance_Cells"        => Phase_Resistance_Cells,
        "Impedance_Stack_Real"          => Impedance_Stack_Real,
        "Impedance_Stack_Imag"          => Impedance_Stack_Imag,
        "Impedance_Cells_Real"          => Impedance_Cells_Real,
        "Impedance_Cells_Imag"          => Impedance_Cells_Imag,
    );
    compress = true)
end

###-----------------------------------------------------------------------------------------------------------
# Generating Plots
#-----------------------------------------------------------------------------------------------------------
#   Read Data from .mat File
#-----------------------------------------------------------------------------------------------------------
fid = matopen(FFT_Results_Datasafe, "r")
    FFT_Stack_U_Ampl            = vec(read(fid, "FFT_Stack_U_Ampl"))
    FFT_Stack_U_Angle           = vec(read(fid, "FFT_Stack_U_Angle"))
    FFT_Stack_U_Freq            = vec(read(fid, "FFT_Stack_U_Freq"))
    FFT_Stack_I_Sum_Ampl        = vec(read(fid, "FFT_Stack_I_Sum_Ampl"))
    FFT_Stack_I_Sum_Angle       = vec(read(fid, "FFT_Stack_I_Sum_Angle"))
    FFT_Stack_I_Sum_Freq        = vec(read(fid, "FFT_Stack_I_Sum_Freq"))
    FFT_Cells_U_Ampl            = Float64.(read(fid, "FFT_Cells_U_Ampl"))
    FFT_Cells_U_Angle           = Float64.(read(fid, "FFT_Cells_U_Angle"))
    FFT_Cells_U_Freq            = Float64.(read(fid, "FFT_Cells_U_Freq"))
    Active_Cell_Area            = Float64(read(fid, "Active_Cell_Area"))
    Cells_per_Segment           = Float64(read(fid, "Cells_per_Segment"))
    Amplitude_Resistance_Stack  = vec(read(fid, "Amplitude_Resistance_Stack"))
    Phase_Resistance_Stack      = vec(read(fid, "Phase_Resistance_Stack"))
    Amplitude_Resistance_Cells  = Float64.(read(fid, "Amplitude_Resistance_Cells"))
    Phase_Resistance_Cells      = Float64.(read(fid, "Phase_Resistance_Cells"))
    Impedance_Stack_Real        = Float64.(read(fid, "Impedance_Stack_Real"))
    Impedance_Stack_Imag        = Float64.(read(fid, "Impedance_Stack_Imag"))
    Impedance_Cells_Real        = Float64.(read(fid, "Impedance_Cells_Real"))
    Impedance_Cells_Imag        = Float64.(read(fid, "Impedance_Cells_Imag"))
close(fid)

###-----------------------------------------------------------------------------------------------------------
# Plot Current over Frequency
#-----------------------------------------------------------------------------------------------------------
_Plot_Path = joinpath(Plots_Path_Current, string("Current_Amplitude_over_Frequency_" * FFT_window_func * Dataformat))
PlotlyJS.savefig(MyLibCurrentoverFrequency(Plotparameter, FFT_Stack_I_Sum_Ampl, FFT_Stack_I_Sum_Freq), _Plot_Path)
```

```
###-------------------------------------------------------------------------------------------------------------------
# Plot Impedance Amplitude Angle Plots / Bode Plot
#-------------------------------------------------------------------------------------------------------------------
# Plot of Stack
_Plot_Path = joinpath(Plots_Path_AmplitudeAngle, string("Impedance_Amplitude_Angle_Plot_Stack_" * FFT_window_func * Dataformat))
PlotlyJS.savefig(MyLibImpedanzAmplitudeAnglePlot(Plotparameter, Amplitude_Resistance_Stack, Phase_Resistance_Stack, FFT_Stack_U_Freq, 0), _Plot_Path)

# Plots for Cells
for cellcounter in eachindex(Amplitude_Resistance_Cells[1, :])
    _Plot_Path = joinpath(Plots_Path_AmplitudeAngle, string("Impedance_Amplitude_Angle_Plot_Cell_" * string(cellcounter) * "_" * FFT_window_func * Dataformat))
    PlotlyJS.savefig(MyLibImpedanzAmplitudeAnglePlot(Plotparameter, Amplitude_Resistance_Cells[:, cellcounter], Phase_Resistance_Cells[:, cellcounter],
                                                     FFT_Cells_U_Freq[cellcounter, :], cellcounter), _Plot_Path)
end

# Plot all Cells in one Plot
_Plot_Path = joinpath(Plots_Path_AmplitudeAngle, string("Impedance_Amplitude_Angle_Plot_Cells_AllinOne_" * FFT_window_func * Dataformat))
PlotlyJS.savefig(MyLibImpedanzAmplitudeAnglePlotAllInOne(Plotparameter, Amplitude_Resistance_Cells, Phase_Resistance_Cells, FFT_Cells_U_Freq[1, :]), _Plot_Path)
# -------------------------------------------------------------------------------------------------------------------


###-------------------------------------------------------------------------------------------------------------------
# Nyquist Plot
#-------------------------------------------------------------------------------------------------------------------
Frequencyvector = Vector{Float64}(undef, 0)
for frequencies in eachindex(FFT_Cells_U_Freq[1, :])
    if FFT_Cells_U_Freq[1, frequencies] <= Border_Freq
        push!(Frequencyvector, FFT_Cells_U_Freq[1, frequencies])
    end
end

# For Stack
_Plot_Path = joinpath(Plots_Path_Nyquist, string("Nyquist_Plot_Stack_" * FFT_window_func * Dataformat))
_Plotdata ,_,_ = MyLibNyquistPlot(Impedance_Stack_Real, Impedance_Stack_Imag, Frequencyvector, _Plot_Parameter = Plotparameter, cellcounter = 0);
PlotlyJS.savefig(_Plotdata, _Plot_Path)

# For Stack 3D, only available as .html
if Plotparameter["Dataformat"] == ".html"
    _Plot_Path = joinpath(Plots_Path_Nyquist, string("Nyquist_Plot_Stack_3D_" * FFT_window_func * Dataformat))
    _Plotdata = MyLibNyquistPlot3D(Plotparameter, Impedance_Stack_Real, Impedance_Stack_Imag, Frequencyvector)
    PlotlyJS.savefig(_Plotdata, _Plot_Path)
end

# For Cells
_HFR_Resistence = Vector{Float64}(undef, 0)
_HFR_Frequency = Vector{Float64}(undef, 0)
for cellcounter in eachindex(Impedance_Cells_Real[1, :])
    _Plot_Path = joinpath(Plots_Path_Nyquist, string("Nyquist_Plot_Cell_" * string(cellcounter) * "_" * FFT_window_func * Dataformat))
    _Plotdata, _HFR_Resistence_temp, _HFR_Frequency_temp = MyLibNyquistPlot(Impedance_Cells_Real[:, cellcounter], Impedance_Cells_Imag[:, cellcounter],
                                                                            Frequencyvector, _Plot_Parameter=Plotparameter, cellcounter=cellcounter)
    PlotlyJS.savefig(_Plotdata, _Plot_Path)
    if Plotparameter["b_HFR_on"]
        push!(_HFR_Resistence, _HFR_Resistence_temp)
        push!(_HFR_Frequency, _HFR_Frequency_temp)
        if cellcounter == lastindex(FFT_Cells_U_Ampl[:, 1])
            _Plot_Path = joinpath(Plots_Path_Nyquist, string("HFR_Plot_Stack_" * FFT_window_func * Dataformat))
            PlotlyJS.savefig(MyLibHFRPlots(Plotparameter, _HFR_Resistence, _HFR_Frequency), _Plot_Path)
        end
    end
end

# For Cells 3D
if Dataformat == ".html"
    for cellcounter in eachindex(Impedance_Cells_Real[1 , :])
        _Plot_Path = joinpath(Plots_Path_Nyquist, string("Nyquist_Plot_Stack_3D_" * string(cellcounter) * "_" * FFT_window_func * Dataformat))
        _Plotdata = MyLibNyquistPlot3D(Plotparameter, Impedance_Cells_Real[:, cellcounter], Impedance_Cells_Imag[:, cellcounter], Frequencyvector, cellcounter)
        PlotlyJS.savefig(_Plotdata, _Plot_Path)
    end
end

# Plot all Cells in one Plot
_Plot_Path = joinpath(Plots_Path_Nyquist, string("Nyquist_Plot_Cells_AllinOne_" * FFT_window_func * Dataformat))
PlotlyJS.savefig(MyLibNyquistPlotAllinOne(Plotparameter, Impedance_Cells_Real, Impedance_Cells_Imag, Frequencyvector), _Plot_Path)
```

MODUL FUNCTIONS FOR STACK EVALUATION

```
# --- ClemensHyCentaFunctionsForStackEvaluation.jl ---------------------------------------------------------------------------------------------
# Collection of functions to analyse harmonic signals in time series data.
#
# Copyright (C) 2022    Stefan N. Pofahl (HyCentA, Graz, Austria)
#
# File Owner:
#   - Clemens Weiskopf
#
# History:
# v1.0 (02-Nov-2022), Clemens Weiskopf
#   - Release of new Version
# v1.1 (13-Dez-2022), Clemens Weiskopf
#   - Changed Nyquist Functioncall to fit updated Version of Plotmodul
# v1.2 (31-Jan-2023), Clemens Weiskopf
#   - Added Function MyLib_Read_Frequency_from_Filename
# ------------------------------------------------------------------------------------------------------------------------------------------------
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
# ------------------------------------------------------------------------------------------------------------------------------------------------
# Markdown notation: https://docs.julialang.org/en/v1/stdlib/Markdown/
# ------------------------------------------------------------------------------------------------------------------------------------------------

module ClemensHyCentaFunctionsForStackEvaluation
using HDF5, Glob, ClemensHyCentaSignalAnalysis
using StatsBase # provides function "mean()" & "std()"
export MyLibCheckandmakeDirectories, MyLibCheckFFTInputParam, MyLibSearchforMeasurementFiles
export MyLibReadHDF5, MyLibCheckPlotParam, MyLib_Read_Frequency_from_Filename

###-----------------------------------------------------------------------------------------------------------------------------------------------
# Informations from E-File
#-----------------------------------------------------------------------------------------------------------------------------------------------
function MyLibReadHDF5(Filepaths)
    if ~isfile(Filepaths)
        error("File \"", Filepaths, "\" not found!")
    end

    # Safe Information as Dict in _data and Convert the Names into Strings
    _data = h5read(Filepaths, "info_Channel_Data")
    _data_names = convert.(String, keys(_data))
    # Count the Segments by Looping through Cellnumbers and compairing them to HDF5 names
    _segment_counter = 0
    for index in eachindex(_data_names)
        # sum is needed because occursin puts out a BitVector, if nothing found beside zeros, its not in the structure
        if sum(occursin.(String("Chan_StackSeg" * string(index) * "_U"), _data_names)) > 0
            _segment_counter = _segment_counter + 1
        end
    end

    # Example for Path: raw"D:\Messdaten\Full-Spec_100A-DC_10A-AC_With_COIL\0013EIS_00p0A000001p39EXC_EIS_20C_EXC_20220804_100255.hdf5"
    # h5open(raw"D:\Messdaten\Full-Spec_100A-DC_10A-AC_With_COIL\0013EIS_00p0A000001p39EXC_EIS_20C_EXC_20220804_100255.hdf5", "r")
    # Safe Data from the fixed Variable Names
    StackSeg_U = Matrix{Float64}(undef, 0, 0)
    fid = h5open(Filepaths, "r")
        Stack_U           = vec(read(fid, "info_Channel_Data/Chan_Stack_U"))
        Stack_I_Sum       = vec(read(fid, "info_Channel_Data/Chan_Stack_I_Sum"))
        Sampling_Freq     = read(fid, "info_Channel_Sampl_Frequ/Chan_Stack_I_Sum")
        Active_Cell_Area = read(fid, "info_Else/Info_Test_UUT_Active_Surface_cm2") # cm2
        # Check if Surface Unit is cm2
        if read(fid, "info_Test/UUT_Active_Surface_Unit") != "cm^2"
            throw(String("Active Surface Unit is not cm^2! Unit of Data is in " * read(fid, "info_Test/UUT_Active_Surface_Unit")))
        end
    close(fid)

    Cells_per_Segment = 1
    # Active_Cell_Area = 75

```

```
    # Loop through Stack Segments
    for loopcounter in 1:_segment_counter
        if loopcounter == 1
            StackSeg_U = vec(_data["Chan_StackSeg1_U"])
        else
            _segment_name = String("Chan_StackSeg" * string(loopcounter) * "_U")
            StackSeg_U = hcat(StackSeg_U, _data[_segment_name])
        end
    end
    StackSeg_U
    return Sampling_Freq, StackSeg_U, Stack_U, Stack_I_Sum, Active_Cell_Area, Cells_per_Segment
end

###-------------------------------------------------------------------------------------------------------------------
# Making and Checking Directories
#-------------------------------------------------------------------------------------------------------------------
function MyLibCheckandmakeDirectories(_FFT_window_func::AbstractString, _Raw_Data_Path::String, _Results_Folder_Path::AbstractString)
    # Check Raw Data Path and Chekc or make Results Path
    if ~isdir(_Results_Folder_Path)
        mkdir(_Results_Folder_Path)
    elseif isempty(_Raw_Data_Path)
        throw("No Raw Data found!")
    end

    # ------------------------------------------------------------------------
    # Check and make Folder for Resultdata \\ResultsFolderPath\\Results
    # ------------------------------------------------------------------------
    Results_Path = joinpath(_Results_Folder_Path, "Results")
    if ~isdir(Results_Path)
        mkdir(Results_Path)
    end

    # ------------------------------------------------------------------------
    # Check and make Folder for Resultplots \\ResultsFolderPath\\Plots
    # ------------------------------------------------------------------------
    Plots_Path = joinpath(_Results_Folder_Path, "Plots")
    if ~isdir(Plots_Path)
        mkdir(Plots_Path)
    end

    # Folders for Plots \\ResultsFolderPath\\Plots\\...
    Plots_Path_AmplitudeAngle = joinpath(Plots_Path, String("Amplitude_Angle_" * _FFT_window_func))
    if ~isdir(Plots_Path_AmplitudeAngle)
        mkdir(Plots_Path_AmplitudeAngle)
    end
    # Folders for Plots \\ResultsFolderPath\\Plots\\...
    Plots_Path_Nyquist = joinpath(Plots_Path, String("Nyquist_" * _FFT_window_func))
    if ~isdir(Plots_Path_Nyquist)
        mkdir(Plots_Path_Nyquist)
    end
    # Folders for Plots \\ResultsFolderPath\\Plots\\...
    Plots_Path_Current = joinpath(Plots_Path, String("Current_" * _FFT_window_func))
    if ~isdir(Plots_Path_Current)
        mkdir(Plots_Path_Current)
    end
    return Results_Path, Plots_Path, Plots_Path_AmplitudeAngle, Plots_Path_Nyquist, Plots_Path_Current
end

###-------------------------------------------------------------------------------------------------------------------
# Only Measurement Files (no Noise Measurement) in Filepath
#-------------------------------------------------------------------------------------------------------------------
function MyLibSearchforMeasurementFiles(Raw_Data_Path::String)
    # Select only Measurement Data Files
    Raw_Data_Path_Vector = Vector{String}(undef, 0)
    Raw_Data_Path = readdir(Raw_Data_Path, join=true)
    for loopcount in eachindex(Raw_Data_Path)
        if occursin("_EXC_", Raw_Data_Path[loopcount])
            push!(Raw_Data_Path_Vector, Raw_Data_Path[loopcount])
        end
    end

    _Sampling_Freq, _, _Stack_U,_, Active_Cell_Area, Cells_per_Segment = MyLibReadHDF5(Raw_Data_Path_Vector[end])

    _freq_Stack_U,_, _amp_Stack_U,_,_,_ = MyLibFFT(vec(_Stack_U .- mean(_Stack_U)), _Sampling_Freq, 0, "flattop", 0)
    _index_ampl_Stack_U = argmax(vec(_amp_Stack_U))
    return Raw_Data_Path_Vector, Active_Cell_Area, Cells_per_Segment, _freq_Stack_U[_index_ampl_Stack_U]
end

###-------------------------------------------------------------------------------------------------------------------
# Checking FFT Input Parameters
#-------------------------------------------------------------------------------------------------------------------
function MyLibCheckFFTInputParam(_FFT_window_func::AbstractString)
    _FFT_window_func = uppercase(_FFT_window_func)
    # Check if Window Function is Correct
    if occursin("HANN", _FFT_window_func)
        _win_func = "hann"
    elseif occursin("HAMMING", _FFT_window_func)
        _win_func = "hamming"
    elseif occursin("KAISER", _FFT_window_func)
        _kaiser_param = filter(isdigit, _FFT_window_func)
        _win_func = String("kaiser" * _kaiser_param)
    elseif occursin("RECT", _FFT_window_func)
        _win_func = "rect"
```

```
    elseif occursin("FLATTOP", _FFT_window_func)
177         _win_func = "flattop"
    elseif occursin("TUKEY", _FFT_window_func)
        _win_func = "tukey"
    elseif occursin("BLACKMANHARRIS", _FFT_window_func)
        _win_func = "blackmanharris"
182 elseif occursin("GAUSS", _FFT_window_func)
        _win_func = "gauss"
    else
        throw("No valid window function present in file name!")
    end
187 end

    ###-------------------------------------------------------------------------------------------------------------------
    # Checking FFT Input Parameters
    #-------------------------------------------------------------------------------------------------------------------
192 function MyLibCheckPlotParam(_Plot_Parameter::Dict)
    # check correct spelling
    if (cmp(uppercase(_Plot_Parameter["Dataformat"]), ".HTML") == 0)
        _Plot_Parameter["Dataformat"] = ".html"
    elseif (cmp(uppercase(_Plot_Parameter["Dataformat"]), ".SVG") == 0)
197         _Plot_Parameter["Dataformat"] = ".svg"
    elseif (cmp(uppercase(_Plot_Parameter["Dataformat"]), ".PDF") == 0)
        _Plot_Parameter["Dataformat"] = ".pdf"
    else
        throw("Dataformat for Plots is not supported or spelled wrong! Supported Formats are: .html; .svg; .pdf")
202 end
    # check if Boolean
    isa(_Plot_Parameter["Transparent_Background"], Bool) == true ? nothing : throw("Transparent_Background must be a Boolean Value!")
    isa(_Plot_Parameter["Meta_Information"], Bool) == true ? nothing : throw("Show_Meta_Information must be a Boolean Value!")
    isa(_Plot_Parameter["Plot_Size"], Bool) == true ? nothing : throw("Plot_Size must be a Boolean Value!")
207 isa(_Plot_Parameter["Transparent_Background_Legend"], Bool) == true ? nothing : throw("Transparent_Background_Legend must be a Boolean Value!")
    isa(_Plot_Parameter["Full_Spectrum"], Bool) == true ? nothing : throw("Full_Spectrum must be a Boolean Value!")
    isa(_Plot_Parameter["Show_Grid"], Bool) == true ? nothing : throw("Show_Grid must be a Boolean Value!")
    return _Plot_Parameter
end
212

function MyLib_Read_Frequency_from_Filename(_Filename::AbstractString)
    position_of_Filename = findlast("\\", _Filename)
    _Filename = _Filename[position_of_Filename[1] : end]
    position_of_A = findfirst("A", _Filename) # find A in Filename -> startingpoint of frequencyinformation
217 _Filename = _Filename[position_of_A[1] + 1 : end] # cut away all characters before A aswell as A
    _Filename = replace(_Filename, "p" => ".", count = 1)# replace the character p with .
    position_of_E = findfirst("E", _Filename) # find E in string for the end of the frequency information
    _Filename = _Filename[1 : position_of_E[1] - 1] # cut away all characters after frequency information
    _Signal_Embossing_Frequency = parse(Float64, _Filename[1 : position_of_E[1] - 1]) # convert string to float
222 return _Signal_Embossing_Frequency
end

end
```

MODUL SIGNAL ANALYSIS

```
# --- HyCentaSignalAnalysis.jl ---------------------------------------------------------------------------------
# Collection of functions to analyse harmonic signals in time series data.
#
# Copyright (C) 2022    Stefan N. Pofahl (HyCentA, Graz, Austria)
#
# File Owner:
#   - Clemens Weiskopf
#
# History:
# v1.0 (25-Okt-2022), Clemens Weiskopf
#   - Created new File and integrated the Trim Function from the Modul StackEvaluation
# ------------------------------------------------------------------------------------------------------------------
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
# ------------------------------------------------------------------------------------------------------------------
# Markdown notation: https://docs.julialang.org/en/v1/stdlib/Markdown/
# ------------------------------------------------------------------------------------------------------------------

module ClemensHyCentaSignalAnalysis

using FFTW
using SpecialFunctions # provides function "besseli()"
using RobustModels #provides "mean()"
using Printf
using WindowFunctions # installation over github url Pkg.add(url="https://github.com/sonosole/WindowFunctions.jl.git")

export MyLibFFT, MyLibInputSignalTrim

# --- Trim Function ----------------------------------------------------------------------------------------------
@doc """
function MyLibTrimInputSignal(_Sampling_Freq::Real, _Frequency::Real, _Signal::Vector{<:Number}, _Trim_of_Signal::Real)

        Checks if the value is <0 or >0 and on the basis of this calculates the upper and lower border

# Inputs:

    • _Sampling_Freq:       ..... Sampling Frequency of Rawdata
    • _Frequency:           ..... Frequency of Signal (extimated from FFT)
    • _Signal:              ..... Data which should be trimmed
    • _Trim_of_Signal       ..... Time in s which get cut away from the beginning of the Signal

# Output:

    • _Trimmed_Signal:      ..... Dataset which is trimmed to full Periods

# Example:

    MyLibTrimInputSignal(_Sampling_Freq, _freq_Stack_U[_index_ampl_Stack_U], _Stack_U, FFT_Trim_of_Signal)

# Current Developer:
    Clemens Weiskopf (25-10-2022)

"""
function MyLibInputSignalTrim(_Sampling_Freq::Real, _Frequency::Real, _Signal::Vector{<:Number}, _Trim_of_Signal::Real)
    # Cut away specific Time from the beginning of the Signal
    _Signal = _Signal[trunc(Int, _Sampling_Freq * _Trim_of_Signal) : end]
    # Calc Quantity of Periods
    _Number_of_Points_per_Period    = Int(_Sampling_Freq / _Frequency)
    _Number_of_Periods              = length(_Signal) / _Number_of_Points_per_Period
    if _Number_of_Periods < 4
        @info String("Number of Periods is under 4; " * " Periods left in Signal: " * string(_Number_of_Periods))
    elseif _Number_of_Points_per_Period < 4
        @info String("Number of Points per Period is under 4; " * " Points per Period: " * string(_Number_of_Points_per_Period))
    end

    # If _Number_of_Periods * _Number_of_Points_per_Period has same Length as _Signal -> avoid index Zero (_Signal[0:end])
```

```
    if _Number_of_Periods * _Number_of_Points_per_Period - 1 == length(_Signal)
        _Trimmed_Signal = _Signal
    else
        _Trimmed_Signal = _Signal[end - (_Number_of_Periods * _Number_of_Points_per_Period - 1) : end]
    end
    # Centering the Signal around the Mean Value
    _Trimmed_Signal = _Trimmed_Signal .- mean(_Trimmed_Signal)
    return _Trimmed_Signal
end

#----- Windonw Function ------------------------------------------------------------------------------------------
@doc """
    _MyLibWindowFunc(_window_type::AbstractString, _window_length::Int)

    Implementation of some established window functions for the purpose of FFT-calculations.
    Intern function called by function "MyLibFFT".

    # Current Developer:
        Clemens Weiskopf (14-07-2022)

"""
function _MyLibWindowFunc(_window_type::AbstractString, _window_length::Int)
    # read Kaiserparameter from String
    if occursin("kaiser", _window_type)
        _kaiser_param = parse(Int64, filter(isdigit, _window_type))
        _window_type = "kaiser"
    end

    window_func = []
    if _window_type == "hann"
        window_func = hanning(_window_length)
    elseif _window_type == "hamming"
        window_func = hamming(_window_length)
    elseif _window_type == "kaiser" # besseli() Modified Bessel function of first kind
        temp = sqrt.(vec(ones(Float32, (_window_length, 1))) - (((0 : (_window_length - 1))-(fill((_window_length - 1) / 2, _window_length))) ./
                    (fill((_window_length - 1) / 2, _window_length))) .^2)
        window_func = besseli.(0, _kaiser_param*temp) / besseli(0, _kaiser_param)
    elseif _window_type == "rect"
        window_func = ones(_window_length, 1);
    elseif _window_type == "flattop"
        window_func = flattop(_window_length)
    elseif _window_type == "tukey"
        window_func = tukey(_window_length)
    elseif _window_type == "blackmanharris"
        window_func = blackmanharris(_window_length)
    elseif _window_type == "gauss"
        window_func = blackmanharris(_window_length)
    else
        window_func = nothing
        error("Specified window-function-type ", _window_type, " not implemented!")
    end
    return vec(window_func)
end

#----- FFT Function ----------------------------------------------------------------------------------------------
@doc """
    function MyLibFFT(_data_vec::Vector{<:Number}, _sampl_rate::Real,
        _window_length::Int=0, _window_type::AbstractString="rect", _overlap::Real=0, _kaiser_param::Real=38)

        Performes the FFT and scales the amplitudes. Only "left" part of FFT spectra is returned,
        without signal at frequency of 0Hz (first element of FFT-output)

# Inputs:

    * _data_vec:    ..... measured data
    * _sampl_rate:  ..... sampling rate in Hz
    optional parameters:
    * _window_length: ... sample of points for one FFT-call,
                          default: window length = sample length (perform one FFT over all points)
    * _window_type: ..... window function (shape data sample), one of: "hann", "hamming", "kaiser", "rect"
                          default: "rect" (without explicit window function)
    * _overlap:  ........ amount of overlap of two consecutive windows, range = [0 .. 0.999]
                          default: 0


# Output:

    * frequncy_vector: .... Frequencies of the FFT bins (discrete frequencies)
    * time_stamps: ........ Time stamp in the middle of one FFT-window (vector-size equal to number of windows)
    * amplitudes: ......... Scaled discrete amplitudes (matrix of n columns, n = n_windows)
    * angles: ............. Angles in radiant/rad (pi rad = 180 degr, matrix of n columns, n = n_windows)
    * FFT_complex: ........ Complex result of FFT (matrix of n columns, n = n_windows)
    * vert_offset: ........ Represents the vertical offset of incoming data sample (scalar value)

# Example:

    frequncy_vector, time_stamps, amplitudes, angles, FFT_complex, vert_offset = MyLibFFT(n_data_vec, sampling_rate)

# Related Internet-Links:
    http://www.fftw.org/

# Current Developer:
    Clemens Weiskopf (14-07-2022)
```

```
"""
function MyLibFFT(_data_vec::Vector{<:Number}, _sampl_rate::Real, _window_length::Int=0, _window_type::AbstractString="rect", _overlap::Real=0)
    _b_DBG = false
    n_data_vec = length(_data_vec)
    if _window_length == 0
        _window_length = n_data_vec
    end
    # ---
    overlap_length = round(_overlap * _window_length)
    if overlap_length >= _window_length
        error("Specified overlap of: ", _overlap," too large!")
    end
    # posible window_types: "hann", "hamming", "kaiser", "rect"
    win_func = _MyLibWindowFunc(_window_type, trunc(Int, _window_length))
    t_win_duration = _window_length / _sampl_rate
    _frequencies = 1 / t_win_duration * (0 : (ceil(Int, _window_length / 2) - 1))
    n_windows = floor(Int, (n_data_vec - _window_length) / (_window_length - overlap_length)) + 1
    # ---
    if n_windows == 1
        _time_stamps = _window_length / 2 / _sampl_rate
    else
        _time_stamps = collect(range(1, step= 1, length= n_windows))
        _time_stamps = _time_stamps .* (_window_length - overlap_length) / _sampl_rate .+ _window_length / 2
    end

    #----- Iteration over the windows ---------------------------------------------------
    _FFTcmplx = [];         # complex fft result vector / matrix if n_windows > 1
    _amplitudes = [];       # scaled fft result vector / matrix if n_windows > 1
    _angles = [];           # angle of complex pointer (vector / matrix if n_windows > 1)
    for i_window in 1:n_windows
        # define step width of window
        move_ind = range(((i_window - 1) * _window_length + 1), step=1, stop=(i_window * _window_length - ((i_window - 1) * overlap_length)) )
        range_ = range(1, step=1, stop=ceil(Int, _window_length / 2))
        if i_window == 1
            _FFTcmplx    = rfft(win_func .* _data_vec[move_ind])
            _FFTcmplx    = _FFTcmplx[range_]# use only half Spectrum for Analysis
            _amplitudes  = (2 / sum(win_func)) .* abs.(_FFTcmplx)
            _angles      = vec(angle.(_FFTcmplx));   # calc angle of complex pointer
        else
            i_FFTcmplx   = fft(win_func .* _data_vec[move_ind])
            i_FFTcmplx   = i_FFTcmplx[range_]
            _FFTcmplx    = hcat(_FFTcmplx, i_FFTcmplx)
            _amplitudes  = hcat(_amplitudes, (2 / sum(win_func)) .* abs.(i_FFTcmplx))
            _angles      = hcat(_angles, angle.(i_FFTcmplx) )
        end
    end
    if _b_DBG
        println("_FFTcmplx, size(): ", size(_FFTcmplx), ", \t typeof(): ", typeof(_FFTcmplx), ", \t size(abs.(_FFTcmplx[1, ])): ", size(abs.(_FFTcmplx[1, ])))
    end
    offset = abs.(_FFTcmplx[1, ])/sum(win_func)
    # --- return values:
    return collect(_frequencies[2:end]), _time_stamps, _amplitudes[2:end, :], _angles[2:end, :], _FFTcmplx[2:end, :], offset
end ## --- end function MyLibFFT ----------------------------------------------------------------------------------------

end ## --- end module ---------------------------------------------------------------------------------------------------
#########################################################################################################################
```

MODUL PLOT FUNCTIONS

```
# --- ClemensHyCentaPlotFunctions.jl ---------------------------------------------------------------------------------
# Collection of functions to analyse harmonic signals in time series data.
#
# Copyright (C) 2022    Stefan N. Pofahl (HyCentA, Graz, Austria)
#
# File Owner:
#   - Clemens Weiskopf
#
# History:
# v1.0 (02-Nov-2022), Clemens Weiskopf
#   - Release of new Version
# v1.0 (15-Dez-2022), Clemens Weiskopf
#   - Added new Functions MyLibNyquistPlotComparison and MyLibImpedanzAmplitudeAnglePlotComparison
# ------------------------------------------------------------------------------------------------------------------
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
# ------------------------------------------------------------------------------------------------------------------
# Markdown notation: https://docs.julialang.org/en/v1/stdlib/Markdown/
# ------------------------------------------------------------------------------------------------------------------
module ClemensHyCentaPlotFunctions
using PlotlyJS, Interpolations, MAT
export MyLibImpedanzAmplitudeAnglePlot, MyLibImpedanzAmplitudeAnglePlotAllInOne, MyLibNyquistPlot, MyLibNyquistPlotAllinOne
export MyLibCurrentoverFrequency, MyLibHFRPlots, MyLibNyquistPlot3D, MyLibImpedanzAmplitudeAnglePlotComparison, MyLibNyquistPlotComparison

@doc """
    MyLibCurrentoverFrequency(_Plot_Parameter::Dict, _Amplitudes::Vector{<:Number}, _Frequencies::Vector{<:Number},
                              _title::AbstractString = "Stack current over Frequency")

        Plotfunction for Current over Frequency Plot

# Inputs:

    * _Plot_Parameter:    .... Plotparameter
    * _Amplitudes:        .... Current data
    * _Frequencies:       .... Frequency data
    * _title:             .... Title of Plot

# Output:

    * _Plotdata:          .... Plotdata for PlotlyJS Plot

# Example:

    PlotlyJS.savefig(MyLibCurrentoverFrequency(Plotparameter, FFT_Stack_I_Sum_Ampl, FFT_Stack_I_Sum_Freq), _Plot_Path)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibCurrentoverFrequency(_Plot_Parameter::Dict, _Amplitudes::Vector{<:Number}, _Frequencies::Vector{<:Number},
                                   _title::AbstractString = "Stack current over Frequency")

    # Frequency Information on plotted Points
    _Frequencies_Plot = Vector{String}(undef, 0)
    for loopcounter in eachindex(_Frequencies)
    _freq_plot_temp = String(string(round(_Frequencies[loopcounter], digits = 2)) * " Hz")
    push!(_Frequencies_Plot, _freq_plot_temp)
    end

    _Plotdata = PlotlyJS.plot(PlotlyJS.scatter(; x = _Frequencies, y = _Amplitudes, name = "Stack current<br>over Frequency",
                                               hovertemplate = "%{y:.2f} A<br>%{x:.2f} Hz", line_width = 2),
    Layout(title_text = _title, xaxis_title_text = "Frequency / Hz", yaxis_title_text = "i / A", xaxis_type = "log",
            yaxis_type = "line", colorway = _MyLib_ColorMap_Plasma(1)))
```

```
    _MyLibPlotlyLayoutSubPlot!(_Plotdata, _Plot_Parameter)
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)
    return _Plotdata
end

@doc """
    MyLibImpedanzAmplitudeAnglePlot(_Plot_Parameter::Dict, _Amplitude_resistance::Vector{<:Number}, _Phase_resistance::Vector{<:Number},
                                    _Frequency::Vector{<:Number}, cellcounter::Integer = 0)

        Plotfunction for Amplitude and Phase over Frequency (Bode Plot)

# Inputs:

    * _Plot_Parameter:        .... Plotparameter
    * _Amplitude_resistance:  .... Amplitude data
    * _Phase_resistance:      .... Phase data
    * _Frequency:             .... Frequency data
    * cellcounter:            .... Number of Cell which is prozessed

# Output:

    * _Plotdata:              .... Plotdata for PlotlyJS Plot

# Example:

    PlotlyJS.savefig(MyLibImpedanzAmplitudeAnglePlot(Plotparameter, Amplitude_Resistance_Stack, Phase_Resistance_Stack, FFT_Stack_U_Freq, 0), _Plot_Path)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibImpedanzAmplitudeAnglePlot(_Plot_Parameter::Dict, _Amplitude_resistance::Vector{<:Number}, _Phase_resistance::Vector{<:Number},
                                         _Frequency::Vector{<:Number}, cellcounter::Integer = 0)

    _Layout_Amplitude = Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "Z / \ohm•cm<sup>2</sup>",
                               xaxis_type = "log", yaxis_type = "line", colorway = _MyLib_ColorMap_Plasma(1))
    _MyLibPlotlyLayoutSubPlot!(_Layout_Amplitude, _Plot_Parameter)
    if cellcounter == 0
        _Plot_Amplitude = PlotlyJS.plot(PlotlyJS.scatter(;x = _Frequency, y = _Amplitude_resistance,
            hovertemplate = "%{y:.2f} \ohm•cm<sup>2</sup><br>%{x:.2f} Hz", name = "Amplitude<br>Stack", line_width = 2)
            , _Layout_Amplitude)
    else
    _Plot_Amplitude = PlotlyJS.plot(PlotlyJS.scatter(;x = _Frequency, y = _Amplitude_resistance,
                hovertemplate = "%{y:.2f} \ohm•cm<sup>2</sup><br>%{x:.2f} Hz", name = "Amplitude<br>Cell " • string(cellcounter), line_width = 2)
                , _Layout_Amplitude)
    end

    _Layout_Phase = Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "\varphi / deg",
                           xaxis_type = "log", yaxis_type = "line", colorway = _MyLib_ColorMap_Plasma(1))
    _MyLibPlotlyLayoutSubPlot!(_Layout_Phase, _Plot_Parameter)
    if cellcounter == 0
        _Plot_Phase = PlotlyJS.plot(PlotlyJS.scatter(;x = _Frequency, y = _Phase_resistance,
            hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz", name = "Phase<br>Stack", line_width = 2), _Layout_Phase)
    else
        _Plot_Phase = PlotlyJS.plot(PlotlyJS.scatter(;x = _Frequency, y = _Phase_resistance,
             hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz", name = "Phase<br>Cell " • string(cellcounter), line_width = 2), _Layout_Phase)
    end

    _Plotdata = [_Plot_Amplitude; _Plot_Phase]

    if cellcounter == 0
        PlotlyJS.relayout!(_Plotdata, title_text = String("Impedance Amplitude and Angle Plot" • " Stack"),
                           showlegend = false, colorway = _MyLib_ColorMap_Plasma(1))
    else
        PlotlyJS.relayout!(_Plotdata, title_text = String("Impedance Amplitude and Angle Plot" • " Cell " • string(cellcounter)),
                           showlegend = false, colorway = _MyLib_ColorMap_Plasma(1))
    end

    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)
    return _Plotdata
end

@doc """
    MyLibImpedanzAmplitudeAnglePlotAllInOne(_Plot_Parameter::Dict, _Amplitude_Resistance_Cells::Matrix{<:Number},
                                            _Phase_Resistance_Cells::Matrix{<:Number}, _Frequency::Vector{<:Number})

        Plotfunction for Amplitude and Phase over Frequency (Bode Plot) for all Cells

# Inputs:

    * _Plot_Parameter:        .... Plotparameter
    * _Amplitude_resistance:  .... Amplitude data
    * _Phase_resistance:      .... Phase data
    * _Frequency:             .... Frequency data

# Output:

    * _Plotdata:              .... Plotdata for PlotlyJS Plot

# Example:
```

```
    PlotlyJS.savefig(MyLibImpedanzAmplitudeAnglePlotAllInOne(Plotparameter, Amplitude_Resistance_Cells,
178                                                          Phase_Resistance_Cells, FFT_Cells_U_Freq[1, :]), _Plot_Path)

# Related Internet-Links:
    https://plotly.com/julia/

183 # Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibImpedanzAmplitudeAnglePlotAllInOne(_Plot_Parameter::Dict, _Amplitude_Resistance_Cells::Matrix{<:Number},
188                                              _Phase_Resistance_Cells::Matrix{<:Number}, _Frequency::Vector{<:Number})

    _Plot_Amplitude_data = AbstractTrace[]
    _Plot_Phase_data     = AbstractTrace[]
    for cellcounter in eachindex(_Amplitude_Resistance_Cells[1, :])
193     line_data_amp = PlotlyJS.scatter(;x = _Frequency, y = _Amplitude_Resistance_Cells[:, cellcounter], name = String("Amplitude Cell " * string(cellcounter)),
                                         hovertemplate = "%{y:.2f} \ohm*cm<sup>2</sup><br>%{x:.2f} Hz<br>Cell " .* string(cellcounter))
        push!(_Plot_Amplitude_data, line_data_amp)

        line_data_ang = PlotlyJS.scatter(;x = _Frequency, y = _Phase_Resistance_Cells[:, cellcounter], name = String("Phase Cell " * string(cellcounter)),
198                                      hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz<br>Cell " .* string(cellcounter))
        push!(_Plot_Phase_data, line_data_ang)
    end

    _Layout_Amplitude = PlotlyJS.Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "Z / \ohm*cm<sup>2</sup>", xaxis_type = "log")
203 _MyLibPlotlyLayoutSubPlot!(_Layout_Amplitude, _Plot_Parameter)
    _Plot_Amplitude = PlotlyJS.plot(_Plot_Amplitude_data, _Layout_Amplitude)

    _Layout_Phase     = PlotlyJS.Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "\varphi / deg", xaxis_type = "log")
    _MyLibPlotlyLayoutSubPlot!(_Layout_Phase, _Plot_Parameter)
208 _Plot_Phase     = PlotlyJS.plot(_Plot_Phase_data, _Layout_Phase)

    _Plotdata = [_Plot_Amplitude; _Plot_Phase]
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)

213 if _Plot_Parameter["Dataformat"] == ".html"
        relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot all Cells", showlegend = true, colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
    else
        relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot all Cells", showlegend = false, colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
    end
218 return _Plotdata
end

@doc """
    MyLibNyquistPlot(_Impedance_Cells_Real::Vector{<:Number}, _Impedance_Cells_Imag::Vector{<:Number}, _Frequencies::Vector{<:Number};
223                  _Plot_Parameter::Dict= Dict(), cellcounter::Integer = 0)

        Plotfunction for Nyquist Plot of Impedance data

# Inputs:
228
    * _Impedance_Cells_Real: .... Real part of impedance data
    * _Impedance_Cells_Imag: .... Imaginary part of impedance data
    * _Frequency:            .... Frequency data
    optional parameters:
233 * _Plot_Parameter:       .... Plotparameter
    * cellcounter:           .... Number of Cell which is prozessed

# Output:

238 * _Plotdata:             .... Plotdata for PlotlyJS Plot

# Example:

    _Plotdata ,_,_ = MyLibNyquistPlot(Impedance_Stack_Real, Impedance_Stack_Imag, Frequencyvector, _Plot_Parameter = Plotparameter, cellcounter = 0)
243 _Plotdata, _HFR_Resistence_temp, _HFR_Frequency_temp = MyLibNyquistPlot(Impedance_Cells_Real[:, cellcounter], Impedance_Cells_Imag[:, cellcounter],
                                                                           Frequencyvector, _Plot_Parameter=Plotparameter, cellcounter=cellcounter)

# Related Internet-Links:
    https://plotly.com/julia/
248
# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
253 function MyLibNyquistPlot(_Impedance_Cells_Real::Vector{<:Number}, _Impedance_Cells_Imag::Vector{<:Number}, _Frequencies::Vector{<:Number};
    _Plot_Parameter::Dict= Dict(), cellcounter::Integer = 0)
    # ---
    if ~haskey(_Plot_Parameter, "Full_Spectrum")
        _Plot_Parameter["Full_Spectrum"] = true
258 end
    if ~haskey(_Plot_Parameter, "Meta_Information")
        _Plot_Parameter["Meta_Information"] = true
    end
    if ~haskey(_Plot_Parameter, "Dataformat")
263     _Plot_Parameter["Dataformat"] = ".html"
    end
    if ~haskey(_Plot_Parameter, "Transparent_Background_Legend")
        _Plot_Parameter["Transparent_Background_Legend"] = true
    end
268 if ~haskey(_Plot_Parameter, "Transparent_Background")
```

```
        _Plot_Parameter["Transparent_Background"] = true
    end
    if ~haskey(_Plot_Parameter, "Show_Grid")
        _Plot_Parameter["Show_Grid"] = true
    end
    if ~haskey(_Plot_Parameter, "Plot_Size")
        _Plot_Parameter["Plot_Size"] = true
    end
    if ~haskey(_Plot_Parameter, "CellArea")
        _Plot_Parameter["CellArea"] = 0
    end
    if ~haskey(_Plot_Parameter, "CellNumber")
        _Plot_Parameter["CellNumber"] = 1
    end
    if ~haskey(_Plot_Parameter, "b_HFR_on")
        _Plot_Parameter["b_HFR_on"] = false
    else
        @info("_Plot_Parameter[\"b_off\"] was specified!")
    end
    # ---

    _Impedance_Cells_Real = _Impedance_Cells_Real[1 : length(_Frequencies)]
    _Impedance_Cells_Imag = _Impedance_Cells_Imag[1 : length(_Frequencies)]

    # define string for axis, hover and HFR for Cells of Stack
    if cellcounter == 0 # for Stack
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>"
    else # for Cells
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup><br>"
    end

    # Calc Position and Frequency of HFR
    @info(" Calc Position and Frequency of HFR")
    if _Plot_Parameter["b_HFR_on"]
        @warn("_MyLibCalcHRF()")
        local _HFR_Position, _HFR_Frequency = _MyLibCalcHRF(_Impedance_Cells_Real, _Impedance_Cells_Imag, _Frequencies, _Plot_Parameter["Full_Spectrum"])
        local _HFR_String = String("HFR at " * string(round(_HFR_Position, digits = 5)) * _title_HFR * string(round(_HFR_Frequency, digits = 2)) * " Hz")
    else
        @info("No HFR-Interpolation!")
        local _HFR_Frequency = nothing
    end

    # Size of tick distance
    _Axis_Extrema_x = extrema(_Impedance_Cells_Real)
    _Axis_Range_x   = (_Axis_Extrema_x[2] - _Axis_Extrema_x[1]) / 10
    _Tick_Distance  = round(_Axis_Range_x, digits = 2)
    if cellcounter == 0
        _Distance_Meta_Text = _Tick_Distance * 1.5
    else
        _Distance_Meta_Text = _Tick_Distance
    end

    # Meta Information -> Decade Frequencies
    if _Plot_Parameter["Meta_Information"] == true
        # Search for Frequencies which mark the Decade transition
        _Frequency_Positions    = Vector{Int}(undef, 0)
        _Frequency_Decade       = 10 .^ (range(0 ,step = 1.0, stop = 5)) # Border Frequencies for every Decade
        _Layout_Annotations     = Vector{PlotlyBase.PlotlyAttribute{Dict{Symbol, Any}}}(undef, 0)

        # find Position of Frequencies
        for loopcounter in eachindex(_Frequency_Decade)
            if isa(findfirst(isone, _Frequencies .> _Frequency_Decade[loopcounter]), Number) == true
                push!(_Frequency_Positions, findfirst(isone, _Frequencies .> _Frequency_Decade[loopcounter]))
            end
        end

        # construct annotations Vector
        for loopcounter in eachindex(_Frequency_Positions)
            _Inplot_Frequencies = string.(round.(_Frequencies[_Frequency_Positions[loopcounter]], digits = 1))
            _Inplot_Text = string(_Inplot_Frequencies * " Hz")
            _x_Position = _Impedance_Cells_Real[_Frequency_Positions[loopcounter]]
            _y_Position = _Impedance_Cells_Imag[_Frequency_Positions[loopcounter]]
            # check if vector index is not exceeding vector
            _Frequency_Position_low = _Frequency_Positions[loopcounter] - 1
            _Frequency_Position_high = _Frequency_Positions[loopcounter] + 1
            if _Frequency_Position_low == 0
                _Frequency_Position_low = 1
            end
            if _Frequency_Position_high > lastindex(_Impedance_Cells_Real)
                _Frequency_Position_high = _Frequency_Position_high - 1
            end
            # Calc the Textposition from the point before and after the Decade Frequency, while also checking if vector index is not exceeded
            _x_Textposition, _y_Textposition = _MyLibCalcHovertextDistance([_Impedance_Cells_Real[_Frequency_Position_low],
                _Impedance_Cells_Real[_Frequency_Position_high]], [_Impedance_Cells_Imag[_Frequency_Position_low],
_Impedance_Cells_Imag[_Frequency_Position_high]]
                , _Distance_Meta_Text)
```

```
            push!(_Layout_Annotations, attr(x = _x_Position, y = _y_Position, text =  _Inplot_Text, showarrow = true, arrowhead=2, bgcolor = "white", font_size=10,
                                        axref = "x", ax = _x_Textposition, ayref = "y", ay = _y_Textposition))
        end
    end

    # Hover Information of Frequency
    _Hover_Frequency = string.(round.(_Frequencies, digits = 1)) .* " Hz"

    _Plot_Lines = AbstractTrace[]
    _Line_Plot = PlotlyJS.scatter(; x = _Impedance_Cells_Real, y = _Impedance_Cells_Imag, mode = "lines+markers", name = "Impedance response<br>Stack",
                                  hovertemplate = _text_Hover .*  _Hover_Frequency)
    push!(_Plot_Lines, _Line_Plot)

    # include HFR into Plot
    if _Plot_Parameter["b_HFR_on"] && _HFR_Frequency != 0 && _HFR_Frequency !== nothing
        _HFR_plot = PlotlyJS.scatter(; x = [_HFR_Position], y = [0], mode = "markers", text = _HFR_String, name = _HFR_String,
                                     hovertemplate = "HFR at %{x:.4f}" * _title_HFR * string(round(_HFR_Frequency, digits = 2)) * " Hz")
        push!(_Plot_Lines, _HFR_plot)
    end

    if _Plot_Parameter["Meta_Information"] == true
        _Layout_Plot = Layout(title_text = "Nyquist Plot", xaxis_title_text = _title_Xaxis, yaxis_title_text = _title_Yaxis, xaxis_dtick = _Tick_Distance,
            yaxis_dtick = _Tick_Distance, yaxis_scaleanchor = "x", yaxis_scaleratio = 1, legend=attr(x = 1, y = 0, yanchor = "bottom", xanchor = "right"),
            annotations = _Layout_Annotations)
    else
        _Layout_Plot = Layout(title_text = "Nyquist Plot", xaxis_title_text = _title_Xaxis, yaxis_title_text = _title_Yaxis, xaxis_dtick = _Tick_Distance,
                              yaxis_dtick = _Tick_Distance, yaxis_scaleanchor = "x", yaxis_scaleratio = 1, legend = attr(x = 1, y = 0, yanchor = "bottom",
                              xanchor = "right"))
    end

    _Plotdata = PlotlyJS.Plot(_Plot_Lines, _Layout_Plot)
    _MyLibPlotlyLayoutSubPlot!(_Plotdata, _Plot_Parameter)

    if cellcounter == 0
        if _Plot_Parameter["b_HFR_on"] && _HFR_Frequency == 0
            @warn String("No Intersetion Point found on Plot: " * "Nyquist Plot" * " Stack")
            relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Stack "), colorway = _MyLib_ColorMap_Plasma(1))
        else
            relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Stack "), colorway = _MyLib_ColorMap_Plasma(2))
        end

    else
        if _HFR_Frequency == 0
            @warn String("No Intersetion Point found on Plot: " * "Nyquist Plot" * " Cell " * string(cellcounter))
            relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Cell " * string(cellcounter)), colorway = _MyLib_ColorMap_Plasma(1))
        else
            relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Cell " * string(cellcounter)), colorway = _MyLib_ColorMap_Plasma(2))
        end

    end
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)
    if _Plot_Parameter["b_HFR_on"]
        return _Plotdata, _HFR_Position, _HFR_Frequency
    else
        _HFR_Position  = nothing
        _HFR_Frequency = nothing
        return _Plotdata, _HFR_Position, _HFR_Frequency
    end
end

@doc """
    MyLibHFRPlots(_Plot_Parameter::Dict, _Resistance::Vector{<:Number}, _Frequency::Vector{<:Number})

        Plotfunction for HFR and their frequencies for all Cells

# Inputs:

    * _Plot_Parameter:   .... Plotparameter
    * _Resistance:       .... Resistance data
    * _Frequency:        .... Frequency data

# Output:

    * _Plotdata:              .... Plotdata for PlotlyJS Plot

# Example:

    PlotlyJS.savefig(MyLibHFRPlots(Plotparameter, _HFR_Resistence, _HFR_Frequency), _Plot_Path)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibHFRPlots(_Plot_Parameter::Dict, _Resistance::Vector{<:Number}, _Frequency::Vector{<:Number})

    _Layout_Resistance = PlotlyJS.Layout(xaxis_title_text = "Cell Number", yaxis_title_text = "HFR / \ohm*cm<sup>2</sup>")
    _MyLibPlotlyLayoutSubPlot!(_Layout_Resistance, _Plot_Parameter)
    _Plot_Resistance = PlotlyJS.plot(PlotlyJS.scatter(;x = collect(range(1, step = 1, lastindex(_Resistance))), y = _Resistance, line_width = 2,
        hovertemplate = "%{y:.2f} \ohm*cm<sup>2</sup><br>Cell Nr. %{x:.0f}", name = "Resisance HFR"), _Layout_Resistance)

```

```
    _Layout_Frequency = Layout(xaxis_title_text = "Cell Number", yaxis_title_text = "Frequency / Hz")
    _MyLibPlotlyLayoutSubPlot!(_Layout_Frequency, _Plot_Parameter)
    _Plot_Frequency  = PlotlyJS.plot(PlotlyJS.scatter(;x = collect(range(1, step = 1, lastindex(_Frequency))), y = _Frequency, line_width = 2,
        hovertemplate = "%{y:.2f} Hz<br>Cell Nr. %{x:.0f}", name = "Frequencies HFR"), _Layout_Frequency)

    _Plotdata = [_Plot_Resistance; _Plot_Frequency]
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)
    relayout!(_Plotdata, title_text = "HFR and Frequency over Cells", showlegend = false, colorway = _MyLib_ColorMap_Plasma(1))
    return _Plotdata
end

@doc """
    MyLibNyquistPlotAllinOne(_Plot_Parameter::Dict, _Impedance_Cells_Real::Matrix{<:Number}, _Impedance_Cells_Imag::Matrix{<:Number},
                             _Frequencies::Vector{<:Number})

        Plotfunction for Nyquist Plot of all Cells

# Inputs:

    * _Plot_Parameter:          .... Plotparameter
    * _Impedance_Cells_Real:    .... Resistance data
    * _Impedance_Cells_Imag:    .... Frequency data
    * _Frequencies:             .... Frequency data

# Output:

    * _Plotdata:                .... Plotdata for PlotlyJS Plot

# Example:

    PlotlyJS.savefig(MyLibNyquistPlotAllinOne(Plotparameter, Impedance_Cells_Real, Impedance_Cells_Imag, Frequencyvector), _Plot_Path)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibNyquistPlotAllinOne(_Plot_Parameter::Dict, _Impedance_Cells_Real::Matrix{<:Number}, _Impedance_Cells_Imag::Matrix{<:Number},
                                  _Frequencies::Vector{<:Number})

    _Frequencies_Plot    = Vector{String}(undef, 0)

    _Plotvector = AbstractTrace[]
    for cellcounter in eachindex(_Impedance_Cells_Real[1, :])
        # Frequency Information on plotted Points
        for loopcounter in eachindex(_Frequencies)
            _freq_plot_temp = String(string(round(_Frequencies[loopcounter], digits = 2)) * " Hz, " * "Cell " * string(cellcounter))
            push!(_Frequencies_Plot, _freq_plot_temp)
        end
        # limit Frequencys to Border Frequency [1 : length(_Frequencies)]
        _line_data = PlotlyJS.scatter(; x = _Impedance_Cells_Real[1 : length(_Frequencies) , cellcounter],
            y = _Impedance_Cells_Imag[1 : length(_Frequencies) , cellcounter], name = String("Cell " * string(cellcounter)), mode = "lines+markers",
            hovertemplate = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup><br>" .* _Frequencies_Plot)
        push!(_Plotvector, _line_data)
        _Frequencies_Plot = []
    end
    _Layout_Plotdata = Layout(title_text = "Nyquist Plot all Cells", xaxis_title_text = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>",
        yaxis_title_text = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>", xaxis_dtick = 0.05, yaxis_scaleanchor = "x", yaxis_scaleratio = 1,
        yaxis_dtick = 0.05, colorway = _MyLib_ColorMap_Plasma(length(_Impedance_Cells_Real[1, :])))
    _MyLibPlotlyLayoutSubPlot!(_Layout_Plotdata, _Plot_Parameter)

    _Plotdata = PlotlyJS.Plot(_Plotvector, _Layout_Plotdata)
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)
    return _Plotdata
end

###-----------------------------------------------------------------------------------------------------------------------
# Plots to Compair differnt Measurements
#-----------------------------------------------------------------------------------------------------------------------
@doc """
    MyLibImpedanzAmplitudeAnglePlotComparison(Filepaths::Vector{String}, Legend_Names::Vector{String}, _Plot_Parameter::Dict; cellcounter::Integer = 0)

        Plotfunction to compare Amplitude and Angle Plots of different measurements

# Inputs:

    * Filepaths:          .... Paths of .mat Files which should be plotted
    * Legend_Names:       .... Legend names of the different data sets
    * _Plot_Parameter:    .... Plotparameter
    * cellcounter:        .... Number of Cell which is prozessed

# Output:

    * _Plotdata:                .... Plotdata for PlotlyJS Plot

# Example:

    _Plotdata = MyLibImpedanzAmplitudeAnglePlotComparison(File_Paths, Legend_Names, Plotparameter, cellcounter = cellcounter)

# Related Internet-Links:
    https://plotly.com/julia/
```

```
# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibImpedanzAmplitudeAnglePlotComparison(Filepaths::Vector{String}, Legend_Names::Vector{String}, _Plot_Parameter::Dict; cellcounter::Integer = 0)
    for bigloopcounter in eachindex(Filepaths)
        # read Data from Files
        if cellcounter == 0
            fid = matopen(Filepaths[bigloopcounter], "r")
                _FFT_Stack_U_Freq                = vec(read(fid, "FFT_Stack_U_Freq"))
                _Amplitude_Resistance_Cells      = vec(read(fid, "Amplitude_Resistance_Stack"))
                _Phase_Resistance_Cells          = vec(read(fid, "Phase_Resistance_Stack"))
            close(fid)
        else
            fid = matopen(Filepaths[bigloopcounter], "r")
                _FFT_Cells_U_Freq                = Float64.(read(fid, "FFT_Cells_U_Freq"))
                _Amplitude_Resistance_Cells      = Float64.(read(fid, "Amplitude_Resistance_Cells"))
                _Phase_Resistance_Cells          = Float64.(read(fid, "Phase_Resistance_Cells"))
            close(fid)
        end

        if bigloopcounter == 1
            _Plot_Amplitude_data = AbstractTrace[]
            _Plot_Phase_data     = AbstractTrace[]
            return _Plot_Amplitude_data, _Plot_Phase_data
        end
        if cellcounter == 0
            line_data_amp = PlotlyJS.scatter(;x = _FFT_Stack_U_Freq, y = _Amplitude_Resistance_Cells, name = String(Legend_Names[bigloopcounter]),
                hovertemplate = "%{y:.2f} \ohm*cm<sup>2</sup><br>%{x:.2f} Hz<br>Stack " * string(Legend_Names[bigloopcounter]))
            push!(_Plot_Amplitude_data, line_data_amp)

            line_data_ang = PlotlyJS.scatter(;x = _FFT_Stack_U_Freq, y = _Phase_Resistance_Cells, name = String(Legend_Names[bigloopcounter]),
                hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz<br>Stack " * string(Legend_Names[bigloopcounter]))
            push!(_Plot_Phase_data, line_data_ang)
        else
            line_data_amp = PlotlyJS.scatter(;x = _FFT_Cells_U_Freq[cellcounter, :], y = _Amplitude_Resistance_Cells[:, cellcounter],
                    name = String(Legend_Names[bigloopcounter]),
                    hovertemplate = "%{y:.2f} \ohm*cm<sup>2</sup><br>%{x:.2f} Hz<br>Cell " .* string(cellcounter) * " " * string(Legend_Names[bigloopcounter]))
            push!(_Plot_Amplitude_data, line_data_amp)

            if bigloopcounter == 2 || bigloopcounter == 10
                line_data_ang = PlotlyJS.scatter(;x = _FFT_Cells_U_Freq[cellcounter, :], y = _Phase_Resistance_Cells[:, cellcounter] .- 360,
                    name = String(Legend_Names[bigloopcounter]),
                    hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz<br>Cell " .* string(cellcounter) * " " * string(Legend_Names[bigloopcounter]))
                push!(_Plot_Phase_data, line_data_ang)
            else
                line_data_ang = PlotlyJS.scatter(;x = _FFT_Cells_U_Freq[cellcounter, :], y = _Phase_Resistance_Cells[:, cellcounter],
                    name = String(Legend_Names[bigloopcounter]),
                    hovertemplate = "%{y:.2f} deg<br>%{x:.2f} Hz<br>Cell " .* string(cellcounter) * " " * string(Legend_Names[bigloopcounter]))
                push!(_Plot_Phase_data, line_data_ang)
            end
        end
    end

    _Layout_Amplitude = PlotlyJS.Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "Z / \ohm*cm<sup>2</sup>", xaxis_type = "log")
    _MyLibPlotlyLayoutSubPlot!(_Layout_Amplitude, _Plot_Parameter)
    _Plot_Amplitude = PlotlyJS.plot(_Plot_Amplitude_data, _Layout_Amplitude)

    _Layout_Phase     = PlotlyJS.Layout(xaxis_title_text = "Frequency / Hz", yaxis_title_text = "\varphi / deg", xaxis_type = "log")
    _MyLibPlotlyLayoutSubPlot!(_Layout_Phase, _Plot_Parameter)
    _Plot_Phase       = PlotlyJS.plot(_Plot_Phase_data, _Layout_Phase)


    _Plotdata = [_Plot_Amplitude; _Plot_Phase]
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)

    if _Plot_Parameter["Dataformat"] == ".html"
        if cellcounter == 0
            relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot Stack", showlegend = true,
                colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
        else
            relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot Cell " * string(cellcounter),
                showlegend = true, colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
        end
    else
        if cellcounter == 0
            relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot Stack", showlegend = false,
                colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
        else
            relayout!(_Plotdata, title_text = "Impedance Amplitude Angle Plot Cell " * string(cellcounter),
                showlegend = false, colorway = _MyLib_ColorMap_Plasma(length(_Plot_Amplitude_data)))
        end
    end
    return _Plotdata
end

@doc """
    MyLibNyquistPlotComparison(Filepaths::Vector{String}, Legend_Names::Vector{String};
                               _Plot_Parameter::Dict= Dict(), cellcounter::Integer = 0, Border_Freq::Integer = 500000)

        Plotfunction to compare Amplitude and Angle Plots of different measurements
```

```
# Inputs:

    - Filepaths:        .... Paths of .mat Files which should be plotted
    - Legend_Names:     .... Legend names of the different data sets
    optional parameters:
    - _Plot_Parameter:  .... Plotparameter
    - cellcounter:      .... Number of Cell which is prozessed
    - Border_Freq:      .... Max. Frequency which is displayed in the Plot

# Output:

    - _Plotdata:                .... Plotdata for PlotlyJS Plot

# Example:

    _Plotdata = MyLibNyquistPlotComparison(File_Paths, Legend_Names, _Plot_Parameter = Plotparameter, cellcounter = cellcounter, Border_Freq = 15000)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)

"""
function MyLibNyquistPlotComparison(Filepaths::Vector{String}, Legend_Names::Vector{String};
    _Plot_Parameter::Dict= Dict(), cellcounter::Integer = 0, Border_Freq::Integer = 500000)
    # ---
    if ~haskey(_Plot_Parameter, "Full_Spectrum")
        _Plot_Parameter["Full_Spectrum"] = true
    end
    if ~haskey(_Plot_Parameter, "Dataformat")
        _Plot_Parameter["Dataformat"] = ".html"
    end
    if ~haskey(_Plot_Parameter, "Transparent_Background_Legend")
        _Plot_Parameter["Transparent_Background_Legend"] = true
    end
    if ~haskey(_Plot_Parameter, "Transparent_Background")
        _Plot_Parameter["Transparent_Background"] = true
    end
    if ~haskey(_Plot_Parameter, "Show_Grid")
        _Plot_Parameter["Show_Grid"] = true
    end
    if ~haskey(_Plot_Parameter, "Plot_Size")
        _Plot_Parameter["Plot_Size"] = true
    end
    if ~haskey(_Plot_Parameter, "CellArea")
        _Plot_Parameter["CellArea"] = 0
    end
    if ~haskey(_Plot_Parameter, "CellNumber")
        _Plot_Parameter["CellNumber"] = 1
    end
    if ~haskey(_Plot_Parameter, "b_HFR_on")
        _Plot_Parameter["b_HFR_on"] = false
    else
        @info("_Plot_Parameter[\"b_off\"] was specified!")
    end
    # ---

    # define string for axis, hover and HFR for Cells of Stack
    if cellcounter == 0 # for Stack
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>"
    else # for Cells
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup><br>"
    end

    for bigloopcounter in eachindex(Filepaths)
        # Read Data from File
        if cellcounter == 0
            fid = matopen(Filepaths[bigloopcounter], "r")
                _FFT_Cells_U_Freq           = vec(read(fid, "FFT_Stack_U_Freq"))
                _Impedance_Stack_Real       = Float64.(read(fid, "Impedance_Stack_Real"))
                _Impedance_Stack_Imag       = Float64.(read(fid, "Impedance_Stack_Imag"))
            close(fid)
        else
            fid = matopen(Filepaths[bigloopcounter], "r")
                _FFT_Cells_U_Freq           = Float64.(read(fid, "FFT_Cells_U_Freq"))
                _Impedance_Cells_Real       = Float64.(read(fid, "Impedance_Cells_Real"))
                _Impedance_Cells_Imag       = Float64.(read(fid, "Impedance_Cells_Imag"))
            close(fid)
        end

        # Limit the Frequencies to certain Value
        Frequencyvector = Vector{Float64}(undef, 0)
        for frequencies in eachindex(_FFT_Cells_U_Freq[cellcounter, :])
            if _FFT_Cells_U_Freq[cellcounter, frequencies] <= Border_Freq
                push!(Frequencyvector, _FFT_Cells_U_Freq[cellcounter, frequencies])
            end
```

```
        end

        if cellcounter == 0
            _Impedance_Cells_Real = _Impedance_Stack_Real[1 : length(Frequencyvector)]
            _Impedance_Cells_Imag = _Impedance_Stack_Imag[1 : length(Frequencyvector)]
        else
            _Impedance_Cells_Real = _Impedance_Cells_Real[1 : length(Frequencyvector), cellcounter]
            _Impedance_Cells_Imag = _Impedance_Cells_Imag[1 : length(Frequencyvector), cellcounter]
        end

        # Calc Position and Frequency of HFR
        @info(" Calc Position and Frequency of HFR")
        if _Plot_Parameter["b_HFR_on"]
            @warn("_MyLibCalcHRF()")
            _HFR_Position, _HFR_Frequency = _MyLibCalcHRF(_Impedance_Cells_Real, _Impedance_Cells_Imag, Frequencyvector, _Plot_Parameter["Full_Spectrum"])
            _HFR_String = String("HFR of " * Legend_Names[bigloopcounter] *  " at " * string(round(_HFR_Position, digits = 5)) *
                                 _title_HFR * string(round(_HFR_Frequency, digits = 2)) * " Hz")
            return _HFR_Position, _HFR_Frequency, _HFR_String
        else
            @info("No HFR-Interpolation!")
            return _HFR_Frequency = nothing
        end

        # Size of tick distance
        _Axis_Extrema_x = extrema(_Impedance_Cells_Real)
        _Axis_Range_x   = (_Axis_Extrema_x[2] - _Axis_Extrema_x[1]) / 10
        _Tick_Distance  = round(_Axis_Range_x, digits = 2)
        if cellcounter == 0
            _Distance_Meta_Text = _Tick_Distance * 1.5
        else
            _Distance_Meta_Text = _Tick_Distance
        end

        # Hover Information of Frequency
        _Hover_Frequency = string.(round.(Frequencyvector, digits = 1)) .* " Hz"
        if bigloopcounter == 1
            _Plot_Lines = AbstractTrace[]
            return _Plot_Lines
        end
        _Line_Plot = PlotlyJS.scatter(; x = _Impedance_Cells_Real, y = _Impedance_Cells_Imag, mode = "lines+markers", name = Legend_Names[bigloopcounter],
                                      hovertemplate = _text_Hover .*  _Hover_Frequency)
        push!(_Plot_Lines, _Line_Plot)

        # include HFR into Plot
        if _Plot_Parameter["b_HFR_on"] && _HFR_Frequency != 0 && _HFR_Frequency !== nothing
            _HFR_plot = PlotlyJS.scatter(; x = [_HFR_Position], y = [0], mode = "markers", text = _HFR_String,
                name = _HFR_String, hovertemplate = "HFR of " * Legend_Names[bigloopcounter] * " at %{x:.4f}" *
                _title_HFR * string(round(_HFR_Frequency, digits = 2)) * " Hz")
            push!(_Plot_Lines, _HFR_plot)
        end

        if bigloopcounter == lastindex(Filepaths)
            _Layout_Plot = Layout(title_text = "Nyquist Plot", xaxis_title_text = _title_Xaxis, yaxis_title_text = _title_Yaxis, xaxis_dtick = _Tick_Distance,
                                  yaxis_dtick = _Tick_Distance, yaxis_scaleanchor = "x", yaxis_scaleratio = 1, legend = attr(x = 1, y = 0, yanchor = "bottom",
                                  xanchor = "right"))
            return _Layout_Plot
        end
    end

    _Plotdata = PlotlyJS.Plot(_Plot_Lines, _Layout_Plot)
    _MyLibPlotlyLayoutSubPlot!(_Plotdata, _Plot_Parameter)

    if cellcounter == 0
        if _Plot_Parameter["b_HFR_on"] && _HFR_Frequency == 0
            @warn String("No Intersetion Point found on Plot: " * "Nyquist Plot" * " Stack")
        end
        relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Stack "), colorway = _MyLib_ColorMap_Plasma(length(_Plot_Lines)))
    else
        if _HFR_Frequency == 0
            @warn String("No Intersetion Point found on Plot: " * "Nyquist Plot" * " Cell " * string(cellcounter))
        end
        relayout!(_Plotdata, title_text = String("Nyquist Plot" * " Cell " * string(cellcounter)), colorway = _MyLib_ColorMap_Plasma(length(_Plot_Lines)))
    end
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)

    return _Plotdata
end

###-----------------------------------------------------------------------------------------------------------------
# 3D Plots
#-----------------------------------------------------------------------------------------------------------------
@doc """
    MyLibNyquistPlot3D(_Plot_Parameter::Dict, _Impedance_Real::Vector{<:Number}, _Impedance_Imag::Vector{<:Number},
                       _Frequencies::Vector{<:Number}, cellcounter::Integer = 0)

        Plotfunction for a 3D Nyquist Plot of a single Measurement

# Inputs:

    * _Plot_Parameter:      .... Plotparameter
    * _Impedance_Real:      .... Real part of impedance data
    * _Impedance_Imag:      .... Imaginary part of impedance data
    * _Frequencies:         .... Frequency data
```

```
    * cellcounter:          .... Number of Cell which is prozessed

# Output:

    * _Plotdata:              .... Plotdata for PlotlyJS Plot

# Example:

    _Plotdata = MyLibNyquistPlot3D(Plotparameter, Impedance_Cells_Real[:, cellcounter], Impedance_Cells_Imag[:, cellcounter], Frequencyvector, cellcounter)

# Related Internet-Links:
    https://plotly.com/julia/

# Current Developer:
    Clemens Weiskopf (02-03-2023)
"""
function MyLibNyquistPlot3D(_Plot_Parameter::Dict, _Impedance_Real::Vector{<:Number}, _Impedance_Imag::Vector{<:Number},
                            _Frequencies::Vector{<:Number}, cellcounter::Integer = 0)

    _Impedance_Real = _Impedance_Real[1 : length(_Frequencies)]
    _Impedance_Imag = _Impedance_Imag[1 : length(_Frequencies)]

    # define string for axis, hover and HFR for Cells of Stack
    if cellcounter == 0 # for Stack
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>*n<sup>-1</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup>*n<sup>-1</sup><br>"
    else # for Cells
        _title_Xaxis = "Z<sub>real</sub> / \ohm*cm<sup>2</sup>"
        _title_Yaxis = "-Z<sub>imag</sub> / \ohm*cm<sup>2</sup>"
        _title_HFR   = " \ohm*cm<sup>2</sup><br>and "
        _text_Hover  = "Z<sub>real</sub>: %{x:.4f} \ohm*cm<sup>2</sup><br>-Z<sub>imag</sub>: %{y:.4f} \ohm*cm<sup>2</sup><br>"
    end

    # Calc Position and Frequency of HFR
    _HFR_Position, _HFR_Frequency = _MyLibCalcHRF(_Impedance_Real, _Impedance_Imag, _Frequencies, _Plot_Parameter["Full_Spectrum"])
    _HFR_String = String("HFR at " * string(round(_HFR_Position, digits = 5)) * _title_HFR * string(round(_HFR_Frequency, digits = 2)) * " Hz")

    # Size of tick distance
    _Axis_Extrema_x = extrema(_Impedance_Real)
    _Axis_Range_x = (_Axis_Extrema_x[2] - _Axis_Extrema_x[1]) / 10
    if cellcounter == 0
        _Tick_Distance = round(_Axis_Range_x, sigdigits = length(string(trunc(Int, _Axis_Range_x))) - 1)
    else
        _Tick_Distance = round(_Axis_Range_x, digits = 2)
    end

    _Plot_Lines = AbstractTrace[]
        _line_plot = PlotlyJS.scatter(x = _Frequencies, y = _Impedance_Real, z = _Impedance_Imag, type = "scatter3d", name = "Impedance Spectrum",
                                      mode = "lines+markers", marker_size = 5, projection_x_show = true, projection_y_show = true,
                                      projection_z_show = true, hovertemplate = _text_Hover)
    push!(_Plot_Lines, _line_plot)

    if _HFR_Frequency != 0
        _HFR_plot = PlotlyJS.scatter(x = [_HFR_Frequency], y = [_HFR_Position], z = [0], mode = "markers",
                                      hovertemplate = "HRF at %{y:.2f}" * _title_HFR * "%{x:.2f} Hz", name = _HFR_String, type = "scatter3d",
                                      marker_size = 5, projection_x_show = true, projection_y_show = true, projection_z_show = true)
        push!(_Plot_Lines, _HFR_plot)
    end


    _Layout_Plot = Layout(margin=attr(l=0, r=0, b=0, t=0), scene = attr(xaxis_title_text = "Frequency / Hz", yaxis_title_text = _title_Xaxis,
                          xaxis_type = "log", zaxis_title_text = _title_Yaxis))
    _MyLibPlotlyLayoutSubPlot!(_Layout_Plot, _Plot_Parameter)

    _Plotdata = PlotlyJS.plot(_Plot_Lines, _Layout_Plot)
    _MyLibPlotlyLayoutMain!(_Plotdata, _Plot_Parameter)

    if cellcounter == 0
        if _HFR_Position == 0 # if there is no intersection point
            @warn String("No Intersetion Point found on Plot: " * "Nyquist 3D Plot" * " Stack")
        end
        relayout!(_Plotdata, title_text = String("Nyquist 3D Plot" * " Stack "))
    else
        if _HFR_Position == 0 # if there is no intersection point
            @warn String("No Intersetion Point found on Plot: " * "Nyquist 3D Plot" * " Cell " * string(cellcounter))
        end
        relayout!(_Plotdata, title_text = String("Nyquist 3D Plot" * " Cell " * string(cellcounter)))
    end
    return _Plotdata
end

###-----------------------------------------------------------------------------------------------------------------
# SubFunctions for Plots
#-----------------------------------------------------------------------------------------------------------------
function _MyLibPlotlyLayoutSubPlot!(_Plotdata::Union{PlotlyJS.SyncPlot, Layout{Dict{Symbol, Any}}, Plot{Vector{AbstractTrace},
                                    Layout{Dict{Symbol, Any}}, Vector{PlotlyFrame}}}, _Plot_Parameter::Dict)
    if ~(cmp(uppercase(_Plot_Parameter["Dataformat"]), ".HTML") == 0)
        PlotlyJS.relayout!(_Plotdata,
            font_family             = "Arial",
            template                = templates.plotly_white,
            height                  = 400,
```

```
            width                   = trunc(Int, sqrt(2) * 400),
            title_x                 = 0.5, # center title
        )
        # Layout Full Size
        if _Plot_Parameter["Plot_Size"] == true
            PlotlyJS.relayout!(_Plotdata,
                xaxis_title_font_size   = 18,
                xaxis_linecolor         = "black",
                xaxis_linewidth         = 2.0,
                xaxis_ticks             = "inside",
                xaxis_tickfont_size     = 14,
                yaxis_title_font_size   = 18,
                yaxis_linecolor         = "black",
                yaxis_linewidth         = 2.0,
                yaxis_ticks             = "inside",
                yaxis_tickfont_size     = 14,
                )
        else # Layout Half Size
            PlotlyJS.relayout!(_Plotdata,
                xaxis_title_font_size   = 28,
                xaxis_linecolor         = "black",
                xaxis_linewidth         = 2.0,
                xaxis_ticks             = "inside",
                xaxis_tickfont_size     = 24,
                yaxis_title_font_size   = 28,
                yaxis_linecolor         = "black",
                yaxis_linewidth         = 2.0,
                yaxis_ticks             = "inside",
                yaxis_tickfont_size     = 24,
                )
        end
        if _Plot_Parameter["Show_Grid"] == true
            PlotlyJS.relayout!(_Plotdata,
                xaxis = attr(gridcolor  = "rgba(160,160,160,100)", showgrid = true, gridwidth = 0.5, zeroline = false, minor_showgrid = true),
                yaxis = attr(gridcolor  = "rgba(160,160,160,100)", showgrid = true, gridwidth = 0.5, zeroline = false,)
            )
        else
            PlotlyJS.relayout!(_Plotdata,
                xaxis_showgrid   = false,
                yaxis_showgrid   = false,
            )
        end
    else
        @info("File extention is \".html\". No relayout done!")
    end
    return _Plotdata
end

function _MyLibPlotlyLayoutMain!(_Plotdata::Union{PlotlyJS.SyncPlot, Plot{Vector{AbstractTrace},
                                 Layout{Dict{Symbol, Any}}, Vector{PlotlyFrame}}}, _Plot_Parameter::Dict)
    if ~(cmp(uppercase(_Plot_Parameter["Dataformat"]), ".HTML") == 0)
        # Layout Full Size
        if _Plot_Parameter["Plot_Size"] == true
            PlotlyJS.relayout!(_Plotdata,
                font_family             = "Arial",
                font_size               = 18,
                legend_font_size        = 14,
                xanchor                 = "center",
                legend_bgcolor          = "white",
            )
        else # Layout Half Size
            PlotlyJS.relayout!(_Plotdata,
                font_family             = "Arial",
                font_size               = 24,
                legend_font_size        = 16,
                xanchor                 = "center",
                legend_bgcolor          = "white",
            )
        end
        # Background of Plot
        if _Plot_Parameter["Transparent_Background"] == true
            PlotlyJS.relayout!(_Plotdata,
                paper_bgcolor   = "rgba(0,0,0,0)", # transparent background
                plot_bgcolor    = "rgba(0,0,0,0)", # transparent background
            )
        else
            PlotlyJS.relayout!(_Plotdata,
                plot_bgcolor    = "white",
            )
        end

        if _Plot_Parameter["Transparent_Background_Legend"] == true
            PlotlyJS.relayout!(_Plotdata,
            legend_bgcolor      = "rgba(0,0,0,0)",
            )
        else
            PlotlyJS.relayout!(_Plotdata,
            legend_bgcolor      = "white",
            legend_bordercolor  = "Black",
            legend_borderwidth  = 1
            )
        end
```

```
    else
        @info("File extention is \".html\". No relayout done!")
    end
    return _Plotdata
end

function _MyLibCalcHovertextDistance(X::Vector{<:Number}, Y::Vector{<:Number}, Length::Number)
    # Calc Slope and offset line
    Slope = ((Y[2] - Y[1]) / (X[2] - X[1]))
    Offset = Y[2] - Slope * X[2]
    # Calc Midpoints of line
    MinMaxValues = extrema(X)
    X_Midpoint =  (MinMaxValues[2] - MinMaxValues[1])/ 2 + minimum(X)
    Y_Midpoint = Slope * X_Midpoint + Offset
    # Calc perpendicular Slope
    PerpSlope = -1 / Slope
    # Calc perpendicular Koordinates so that are always inside of the curve
    if Y[2] > Y[1]
        X_Koordinate = X_Midpoint + (-Length * sqrt(1 / (1 + PerpSlope ^ 2)))
        Y_Koordinate = Y_Midpoint + (-Length * PerpSlope * sqrt(1 / (1 + PerpSlope ^ 2)))
    else
        X_Koordinate = X_Midpoint + (Length * sqrt(1 / (1 + PerpSlope ^ 2)))
        Y_Koordinate = Y_Midpoint + (Length * PerpSlope * sqrt(1 / (1 + PerpSlope ^ 2)))
    end
    return X_Koordinate, Y_Koordinate
end

function _MyLibCalcHRF(_Real::Vector{<:Number}, _Imag::Vector{<:Number}, _Frequencies::Vector{<:Number}, _Full_Spectrum::Bool)

    # Find Position of first negative Point
    _First_Negativ_Point_Pos = []
    if _Full_Spectrum == true
        _loopcounter_HFR = trunc(Int, lastindex(_Imag) / 2)
    else
        _loopcounter_HFR = 1
    end
    for _loopcounter_HFR in _loopcounter_HFR : lastindex(_Imag)
        if _Imag[_loopcounter_HFR] <= 0
            _First_Negativ_Point_Pos = _loopcounter_HFR
            break
        else
            _First_Negativ_Point_Pos = 0
        end
    end

    # Interpolate Position and Frequency of HRF if there is a zeroline crossing
    if _First_Negativ_Point_Pos == 0 ||  sum(_Real) <= 0.0# Check if first Point of Nyquist is negative
        _x_Coordinate = _First_Negativ_Point_Pos
        _HFR_Frequency = 0
    else
        # Interpolation of Position
        _Interpolate_Linear = LinearInterpolation([_Imag[_First_Negativ_Point_Pos], _Imag[_First_Negativ_Point_Pos - 1]],
                                                  [_Real[_First_Negativ_Point_Pos], _Real[_First_Negativ_Point_Pos - 1]])
        _x_Coordinate = sqrt(_Interpolate_Linear(0) ^ 2)

        # Interpolate HRF Frequency
        _Upper_Point_x = sqrt(_Real[_First_Negativ_Point_Pos - 1] ^ 2)
        _Lower_Point_x = sqrt(_Real[_First_Negativ_Point_Pos] ^ 2)
        if _Upper_Point_x  >= _Lower_Point_x
            _x_Distance_Upper_Point = sqrt(_Imag[_First_Negativ_Point_Pos - 1] ^ 2 + (_Upper_Point_x - _x_Coordinate)^2 ) * (-1)
            _x_Distance_Lower_Point = sqrt(_Imag[_First_Negativ_Point_Pos] ^ 2 + (_Lower_Point_x - _x_Coordinate)^2 )

            _Interpolate_Frequency = LinearInterpolation([_x_Distance_Upper_Point, _x_Distance_Lower_Point],
                                                         [_Frequencies[_First_Negativ_Point_Pos - 1], _Frequencies[_First_Negativ_Point_Pos]])
            _HFR_Frequency = _Interpolate_Frequency(0)
        else
            _x_Distance_Upper_Point = sqrt(_Imag[_First_Negativ_Point_Pos - 1] ^ 2 + (_Upper_Point_x - _x_Coordinate)^2 )
            _x_Distance_Lower_Point = sqrt(_Imag[_First_Negativ_Point_Pos] ^ 2 + (_Lower_Point_x - _x_Coordinate)^2 ) * (-1)

            _Interpolate_Frequency = LinearInterpolation([_x_Distance_Lower_Point, _x_Distance_Upper_Point],
                                                         [_Frequencies[_First_Negativ_Point_Pos], _Frequencies[_First_Negativ_Point_Pos - 1]])
            _HFR_Frequency = _Interpolate_Frequency(0)
        end
    end
    return _x_Coordinate, _HFR_Frequency
end

function _MyLib_ColorMap_Plasma(Number_of_Lines::Number)
    color_left   = [250, 219, 36]
    color_middle = [202, 70, 119]
    color_right  = [13, 8, 135]

    if Number_of_Lines == 2
        color_string = ["rgb(13,8,135)","rgb(250,219,36)"]
    else
        range_red_left  = (color_middle[1] - color_left[1]) / (Number_of_Lines / 2)
        range_red_right = (color_right[1] - color_middle[1]) / (Number_of_Lines / 2)

        range_green_left = (color_middle[2] - color_left[2]) / (Number_of_Lines / 2)
        range_green_right = (color_right[2] - color_middle[2]) / (Number_of_Lines / 2)

        range_blue_left = (color_middle[3] - color_left[3]) / (Number_of_Lines / 2)
        range_blue_right = (color_right[3] - color_middle[3]) / (Number_of_Lines / 2)
```

```
            portion_red = Vector{Int64}(undef, 0)
            portion_green = Vector{Int64}(undef, 0)
1108        portion_blue = Vector{Int64}(undef, 0)

            for loopcounter in 1:Number_of_Lines
                if loopcounter <= trunc(Number_of_Lines / 2)
                    if loopcounter == 1
1113                    push!(portion_red, color_left[1])
                        push!(portion_green, color_left[2])
                        push!(portion_blue, color_left[3])
                    else
                        push!(portion_red, trunc(color_left[1] + range_red_left * loopcounter))
1118                    push!(portion_green, trunc(color_left[2] + range_green_left * loopcounter))
                        push!(portion_blue, trunc(color_left[3] + range_blue_left * loopcounter))
                    end
                else
                    if loopcounter == Number_of_Lines
1123                    push!(portion_red, color_right[1])
                        push!(portion_green, color_right[2])
                        push!(portion_blue, color_right[3])
                    else
                        push!(portion_red, trunc(color_middle[1] + range_red_right * (loopcounter - trunc(Number_of_Lines / 2))))
1128                    push!(portion_green, trunc(color_middle[2] + range_green_right * (loopcounter - trunc(Number_of_Lines / 2))))
                        push!(portion_blue, trunc(color_middle[3] + range_blue_right * (loopcounter - trunc(Number_of_Lines / 2))))
                    end
                end
            end
1133
            color_string = Vector{String}(undef, 0)
            for loopcounter in 1:Number_of_Lines
                push!(color_string, String("rgb(" * string(portion_red[loopcounter]) * "," * string(portion_green[loopcounter]) *
                                           "," * string(portion_blue[loopcounter]) * ")"))
1138        end
        end
        return color_string
    end

1143 end
```