Validierung von *Holomorphic Embedding Load Flow*

Benedikt Schmidt (benediktibk@aon.at)

20. August 2014

Inhaltsverzeichnis

1 Einführung **3**

2 Berechnung der Knotenspannungen in *HELM* **3**

3 Implementierung **4**
3.1 Datenbank 6
3.2 Bedienung 8

4 Ergebnisse **9**

5 Fazit **15**

1 Einführung

Klassische Verfahren zur Berechnung von Lastflüssen in elektrischen Energieverteilungsnetzen, wie zum Beispiel die *Stromiteration* oder *Fast-decoupled-load-flow* (*FDLF*), weisen erhebliche Probleme in Bezug auf die Konvergenz auf. Zum einen können diese iterative Verfahren unter Umständen gar nicht konvergieren, obwohl das zu berechnende System an sich stabil wäre. Zusätzlich dazu können die berechneten Lösungen in bestimmten Situationen keine physikalische Lösung des Systems beschreiben. *Holomorphic embedding load flow* (*HELM*) [1], ein von *Antonio Trias* neu entwickelter Ansatz zur Lastflussberechnung, verspricht diese Probleme zu lösen. Die praktische Valdidierung dieser Aussagen erfolgte von mir durch die Entwicklung eines Programmes, welches sowohl *HELM* als auch die klassischen iterativen Verfahren implementiert.

2 Berechnung der Knotenspannungen in *HELM*

Ausgangsproblem ist ein elektrisches Energieverteilungsnetz bestehend aus Knoten bzw. sogenannten Bussen und Admittanzen Y_{ij}, welche die Knoten verbinden. Zudem muss an mindestens einem Knoten, dem Slackbus, Real- und Imaginärteil der Spannung zur Definition des Winkels vorgegeben sein. An allen anderen Knoten müssen dann jeweils genau zwei der folgenden Größen vorgegeben sein: P_i, Q_i, $\Re\{U_i\}$, $\Im\{U_i\}$ oder $|U_i|$. Bei einer Vorgabe von $|U_i|$ und P_i spricht man von einem PV-Bus, bei P_i und Q_i von einem PQ-Bus. Im folgenden werde ich mich immer auf den Fall eines PQ-Busses beziehen, die Modellierung eines PV-Busses in *HELM* wird in [2] behandelt. Die im weiteren Verlauf angeführten Formeln und Methoden entstammen größtenteils [1], [3] und [4].

Für einen PQ-Bus ergibt sich aus der Kirchhoffschen Knotenregel

$$\sum_j Y_{ij} U_j = I_j + \frac{S_j^\star}{U_j^\star}, \tag{1}$$

wobei die konstanten Ströme I_j der Reduktion des Gesamtsystems um die Slackbusse entstammen.

Die explizite Lösung von (1) ist an dieser Stelle aufgrund der quadratischen Form, gemischt mit der Konjugation der Knotenspannungen, nicht möglich. Aus diesem Grund heraus wird in [4] eine Betrachtung der Spannungen als komplexe Funktionen in einem neuen Parameter s vorgeschlagen. Um zudem

zu erreichen, dass diese Funktionen holomporph sind, wird (1) zu

$$\sum_j Y_{ij} U_j(s) = sI_j + \frac{sS_j^\star}{U_j^\star} + (1-s)\sum_j Y_{ij} \tag{2}$$

erweitert. Eine Auswertung dieser Gleichung an der Stelle $s = 1$ ergibt wiederum (1), womit also über $U_i(s = 1)$ die eigentliche Knotenspannung U_i berechnet werden kann. Das Ziel ist somit nicht mehr direkt die Berechnung von U_i, sondern anstelle dessen die Darstellung von $U_i(s)$. Um dies zu erreichen wird $U_i(s)$ durch die reduzierte Laurentreihe

$$U_i(s) = \sum_{n=0}^{\infty} c_{i,n} s^n \tag{3}$$

in (2) ersetzt. Damit lassen sich dann durch einen Koeffizientenvergleich Formeln für die sukzessive Berechnung von $c_{i,n}$ ermitteln.

Die direkte Auswertung von (3) ist an diesem Punkt meist nicht möglich, da die Reihe an der Stelle 0 entwickelt wurde und der Konvergenzradius in den meisten Fällen erheblich kleiner als 1 ist. Dieses Problem lässt sich allerdings über eine Methode zur analytischen Fortsetzung lösen, zum Beispiel mithilfe des Epsilon Algorithmuses von Wynn [5].

Theoretisch lässt sich nun für die Lösung nachweisen, dass sie genau dann konvergiert wenn das System stabil ist, also kein Spannungszusammenbruch eintritt. Somit erhält man also an und für sich genau jenes Ergebnis, welches man sich wünscht: Wenn im System die Spannung zusammenbricht konvergiert das Verfahren nicht, falls das System stabil ist konvergiert das Verfahren. Praktisch ist man allerdings nicht in der Lage zum einen beliebig viele Koeffizienten zu berechnen, und zum anderen stößt man an die Grenzen der Rechengenauigkeit. Bereits für sehr einfache Systeme erreichen die Koeffizienten $c_{i,50}$ dermaßen große Werte, dass mit einem 64 Bit Gleitkommadatentyp die Rechenfehler größer als etwaige Verbesserungen in der Genauigkeit des Ergebnisses sind. Auf Ansätze um dieses Problem anzugehen werde ich im nächsten Abschnitt eingehen.

3 Implementierung

Um *HELM*, bzw. genauer gesagt meine Implementierung von *HELM*, mit den klassischen Ansätzen zur Lastflussberechung vergleichen zu können habe ich folgende Berechnungsverfahren implementiert:

1. *Knotenpunktpotentialverfahren*

2. *Stromiteration*

3. *Newton-Raphson* (ohne Vereinfachung)

4. *FDLF*

5. *HELM* mit doppelter Genauigkeit (64 Bit)

6. *HELM* mit beliebiger Genauigkeit

7. *HELM* kombiniert mit *Newton-Raphson*

8. *HELM* kombiniert mit einer *Stromiteration*

Die zwei unterschiedlichen Varianten von *HELM* ergeben sich aus der Einschränkung eines 64 Bit Gleitkommadatentyps, in den meisten Programmiersprachen bekannt als double. Da für eine größere Genauigkeit mehr als nur 50 Koeffizienten und somit auch eine größerer Datentyp nötig ist, habe ich diesen Teil, als einzigen, in *C++* entwickelt. Dadurch konnte ich auf Templates zurückgreifen um beliebige Datentypen und darauf basierend Matrizen zu verwenden. Für den Gleitkommadatentyp mit beliebiger Genauigkeit kam *MPIR* [1] zum Einsatz, für die Template-Matrizenrechnung *Eigen* [2]. Somit sind die einzigen Einschränkungen für die Rechengenauigkeit der verfügbare Arbeitspeicher und die Berechnungsdauer. Insbesondere letztere wird stark verschlechtert durch die Wahl eines größeren Datentyps, dementsprechend ist dieses Berechnungsverfahren vermutlich eher für akademische Zwecke sinnvoll.

Praktisch durchaus nützlich sein könnten die beiden letzten implementierten Berechnungsverfahren in denen jeweils *HELM* mit doppelter Genaugikeit mit einem iterativen Verfahren kombiniert wird. Bei diesen Kombinationen wird zuerst das iterative Verfahren angewandt, da dieses in der Regel um Größenordnungen schneller ist als *HELM*. Falls allerdings das iterative Verfahren nicht konvergiert kommt *HELM* mit doppelter Genauigkeit zum Einsatz und liefert bessere Startwerte, basierend auf denen wiederum das iterative Verfahren angewandt wird. Somit erhält man gewissermaßen die Vorteile beider Welten, die Geschwindigkeit der iterativen Verfahren kombiniert mit der verbesserten Konvergenz bei Situationen nahe dem Spannungszusammenbruch.

Die Software gliedert sich in mehrere Klassenbibliotheken, welche hierarchisch aufeinander aufbauen. Die wichtigsten Teile zusammengefasst finden sich in Abb. 1. Die Implementierung von *HELM* ist dabei als einziger Teil in

[1] http://www.mpir.org/

[2] http://eigen.tuxfamily.org/index.php

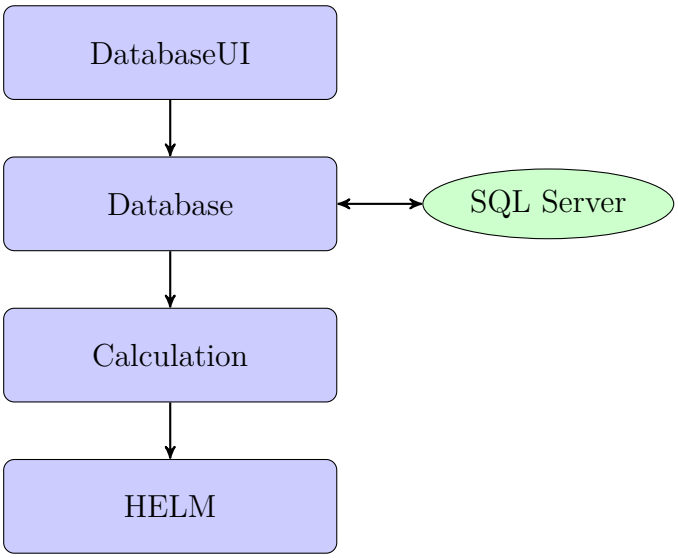

Abbildung 1: Software Architektur

C++, der Rest ist in *C#* implementiert. Außerdem gibt es für die meisten Bibliotheken ein Pendant mit Unittests, welche mit Hilfe des in *Visual Studio* integrierten Testframeworks ausgeführt werden können.

3.1 Datenbank

Im Hintergrund zur eigentlichen Anwendung ist es nötig einen Datenbankserver zu betreiben, idealerweise eine *Microsoft SQL Server* oder *Microsoft SQL Server Express*. Die Informationen über das Netz, also die Tabellen powernets, nodes, loads, transformers, lines, generators und feedins, werden beim Verbinden mit der Datenbank eingelesen. Dementsprechend ist es möglich ohne Verwendung der grafischen Oberfläche direkt über die Datenbank Netze zu definieren. Das dafür zu beachtende Datenbankschema ist in Abb. 2 dargestellt. Die Auswahl des Berechnungsverfahrens erfolgt über die Spalte CalculatorSelection in powernets, die jeweilige Bedeutung der Werte kann Tab. 1 entnommen werden.

Die Tabellen admittances, admittancenodenames und admittancevalues werden nicht eingelesen und dienen nur der Speicherung der Admittanzmatrix, falls der Benutzer dies wünscht. Sowohl in diesen Tabellen als auch in jenen, in denen die Netze abgebildet werden, sind sämtliche Größen in SI-Einheiten.

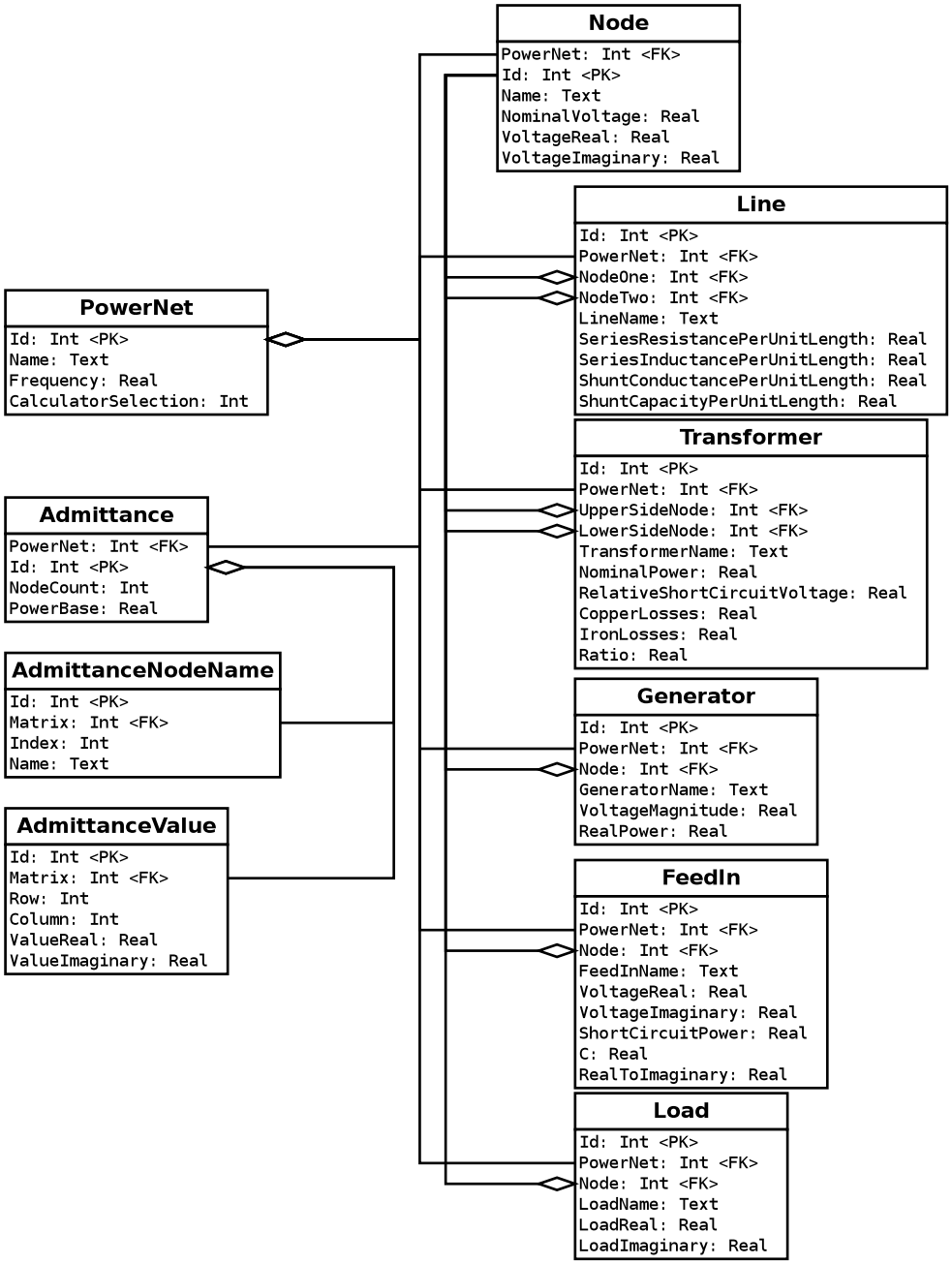

Abbildung 2: Datenbankschema

CalculatorSelection	Berechnungsverfahren
0	*Knotenpunktpotentialverfahren*
1	*Stromiteration*
2	*Newton-Raphson*
3	*FDLF*
4	*HELM*, 64 Bit
5	*HELM*, 200 Bit
6	*HELM* mit *Stromiteration*
7	*HELM* mit *Newton-Raphson*

Tabelle 1: Bedeutung der Werte für CalculatorSelection

3.2 Bedienung

Zum Start der Anwendung muss die Datei *DatabaseUI.exe* ausgeführt werden, welche sich im Build-Verzeichnis des Projektes *DatabaseUI* befindet. Falls diese Anwendung von einem anderen Verzeichnis aus gestartet werden sollte ist es nötig sämtliche Libraries, welche sich im selben Pfad befinden, mit zu kopieren.

Nach Start der Anwendung muss eine Verbindung zur Datenbank aufgebaut werden. Mit den standardmäßig ausgefüllten Werten wird eine Verbindung zu einer lokal gestarteten Instanz eines *Microsoft SQL Express* aufgebaut mit der aktuellen Benutzerkennung. Falls die Felder für den Benutzernamen und das Passwort ausgefüllt werden erfolgt die Authentifikation am Server darüber.

Falls das Datenbankschema im angegebenen Server noch nicht existiert werden die Tabellen bei Bedarf automatisch angelegt. Dafür ist es dann aber nötig, dass der Benutzer die Berechtigungen dazu im SQL Server hat.

Nach einem Aufbau der Verbindung, und unter Umständen dem Anlegen der Tabellen, werden sämtlich Werte, welche sich derzeit im Server befinden, eingelesen. Dieser Schritt erfolgt nur nach dem Aufbau der Verbindung. Falls also im Hintergrund manuell Daten in die Tabellen eingetragen wurden muss die Verbindung getrennt und wieder neu aufgebaut werden um diese Änderungen in der Benutzeroberfläche dargestellt zu bekommen.

Das Eintragen von neuen Element erfolgt an und für sich wie in einer Tabellenkalkulation, die Daten werden automatisch im Hintergrund zum SQL Server übertragen. Sämtliche Größen sind dabei wiederum in SI-Einheiten angegeben. Die Berechnung der Knotenspannungen erfolgt dann über den Button *calculate node voltages*. Die berechneten Spannungen werden in der Tabelle *Nodes* abgespeichert. Es gilt allerdings auch die Ausgabe links unten zu beachten. Falls es nicht möglich war die Spannungen zu berechnen

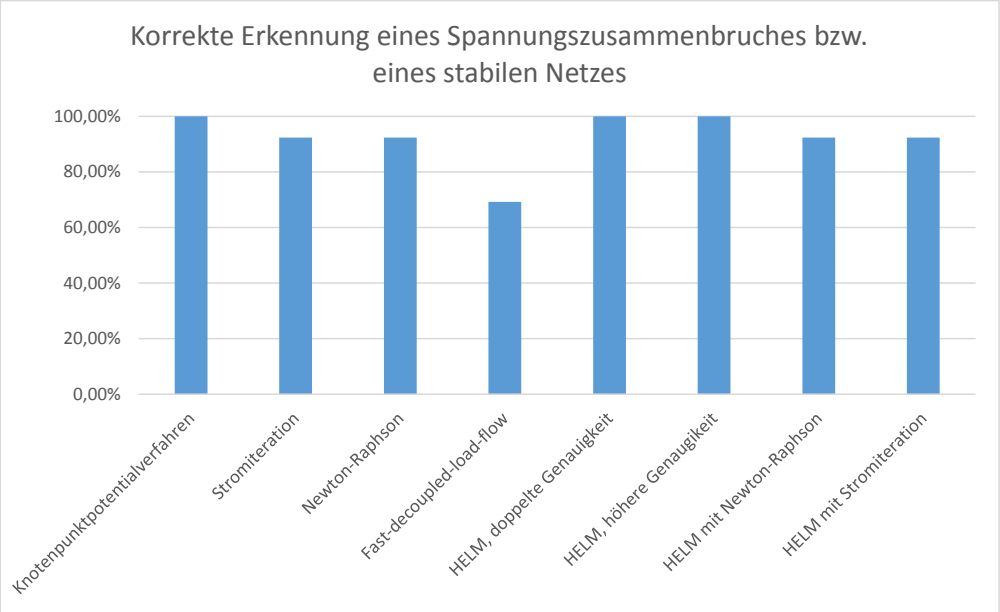

Abbildung 3: Konvergenzverhalten der Berechnungsverfahren

wird hier eine Fehlermeldung ausgegeben.

Zudem ist es noch möglich über den Button *calculate admittance matrix* für das aktuell ausgewählte Netz die Admittanzmatrix in der Datenbank abzulegen.

4 Ergebnisse

Um die unterschiedlichen Berechnungsverfahren möglichst gut vergleichen zu können habe ich eine weitere Benutzeroberfläche in *CalculationComparison* implementiert. Diese ermöglicht alle implementierten Berechnungsverfahren anhand einiger relativ zufällig gewählter Netze zu vergleichen bezüglich Genauigkeit, Konvergenzverhalten und Geschwindigkeit. Beginnen möchte ich hier mit Abb. 3, welche die Anzahl der Netze darstellt, in welchen das jeweilige Verfahren korrekt konvergiert. In diesem Fall kommt es nicht zu der Situation einer Konvergenz zu einem instabilen Arbeitspunkt, daher ist nur die Information darüber ob das Verfahren konvergiert oder nicht relevant. Einige der Beispielnetze sind auch bewusst so gewählt, dass sich ein Spannungszusammenbruch ergibt. In diesen Fällen ist natürlich dann ein korrektes Konvergenzverhalten gegeben wenn das Verfahren nicht konvergiert und somit den Spannungszusammenbruch detektiert. Die Größe des Datentyps ist in *HELM* mit höherer Genauigkeit dabei jeweils so konfiguriert, dass von der

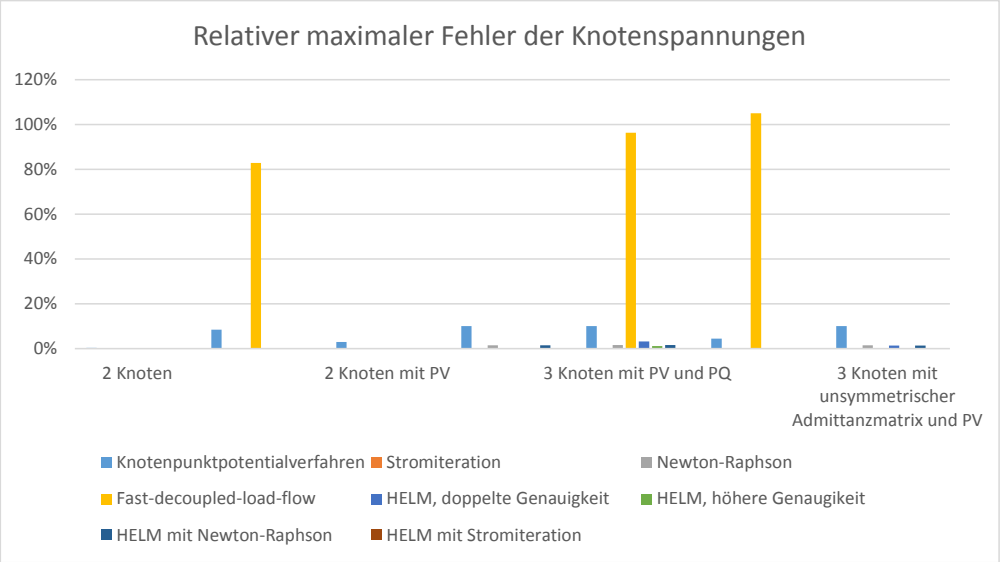

Abbildung 4: Genauigkeit der Berechnungsverfahren

Genauigkeit her mit den iterativen Verfahren vergleichbare Ergebnisse erreicht werden. Exakt gleiche Ergebnisse sind natürlich nicht möglich, ein paar Verfahren konvergieren gar nicht für alle Testnetze korrekt. Allerdings konnte ich mit den standardmäßig angepassten Parametern vergleichbare Resultate bezüglich der Genauigkeit erreichen. Um dies mit *HELM* zu erreichen ist es teilweise nötig einen Datentyp mit mehreren tausend Bit Genauigkeit zu wählen. Außerdem sei noch zu erwähnen, dass die Varianten *HELM* mit *Stromiteration* und *HELM* mit *Newton-Raphson* sich ein klein wenig von denen in Abschnitt 3 unterscheiden. In diesem Fall wird immer zuerst *HELM* mit 64 Bit zur Ermittlung der Startwerte ausgeführt und von diesen Werten aus das jeweilige iterative Verfahren gestartet. Somit stellen die beiden Verfahren in diesem Abschnitt nur einen Zwischenschritt der Verfahren in Abschnitt 3 dar.

Als nächsten Aspekt gehe ich nun auf die Genauigkeit der Verfahren ein, welche durchaus variiert. Hierfür findet sich in Abb. 4 und Abb. 5 nur eine reduzierte Auswahl der Testnetzen von Abb. 3, nämlich auf jene, bei denen alle Verfahren zumindest konvergieren. In Abb. 4 zeigt sich allerdings, dass das *Knotenpunktpotentialverfahren* und *FDLF* teilweise trotzdem nicht wirklich brauchbare Ergebnisse liefern. In Abb. 5 findet sich deswegen noch einmal der selbe Durchlauf, allerdings ohne die beiden zuletzt genannten Verfahren. In diesem Diagramm ist gut erkennbar, dass *HELM* mit nur 64 Bit Rechengenauigkeit in manchen Fällen noch keine ausreichend exakten Ergebnisse ausgibt. Die anderen Verfahren hingegen zeigen ausreichend kleine und ver-

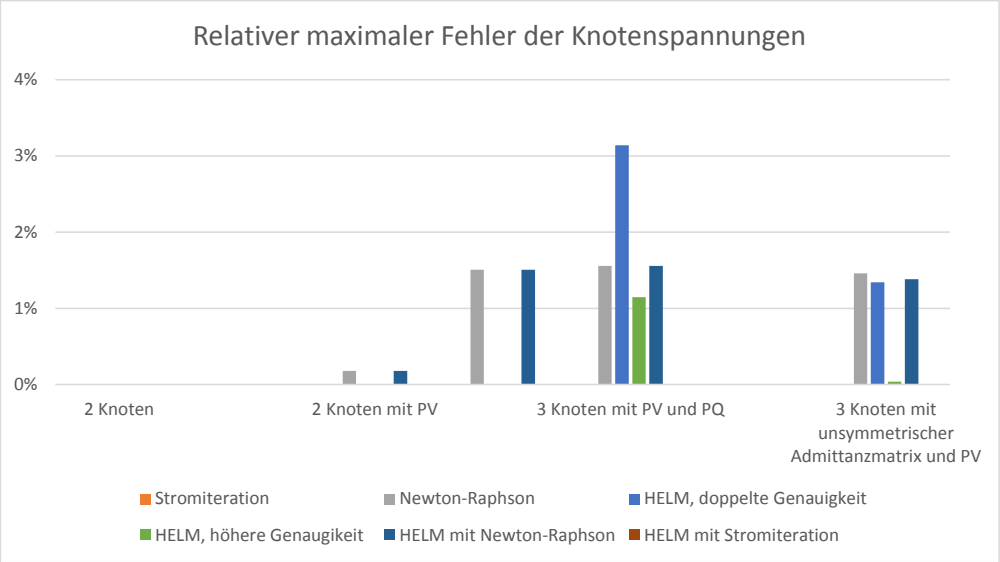

Abbildung 5: Genauigkeit der Berechnungsverfahren

gleichbare Fehler um einen Blick auf die Berechnungsdauer werfen zu können.

Zu guter Letzt stellt sich noch die Frage der Berechnungsdauer, welche insbesondere bei größeren Netzen ein relevanter Aspekt wird. Hierbei erkennt man in Abb. 6 den Nachteil der höheren Rechengenauigkeit von *HELM* mit einem genaueren Datentyp. Dementsprechend vermute ich vor allem akademische Anwendungsgebiete für *HELM* mit beliebiger Genauigkeit. Der Vollständigkeit halber sind in Abb. 7 noch die Schätzungen der Standardabweichungen angegeben welche deutlich zeigen, dass es sich in Abb. 6 um keine einzelnen Ausreißer handelt.

HELM ist aber ein durchaus interessanter Ansatz für Netze, welche nahe am Spannungszusammenbruch betrieben bzw. berechnet werden. Um dieses mögliche Einsatzgebiet zu überprüfen habe ich in dem kleinen Netz in Abb. 10 mit nur einer Einpeisung, einer Last und einer Verbindung dazwischen sukzessive die Last erhöht bis das jeweilige Berechnungsverfahren einen Spannungszusammenbruch detektierte bzw. keine ausreichende Genauigkeit mehr gegeben war. Aufgrund der einfachen Struktur des Netzes lässt sich zudem rechnerisch zeigen, dass die tatsächliche Stabilitätsgrenze bei $P = 0.25\,\mathrm{W}$ liegt. In Abb. 8 und Abb. 9 zeigen sich die besseren Eigenschaften von *HELM*, verglichen mit den klassischen iterativen Verfahren, bei der Berechnung eines Netzes nahe am Spannungszusammenbruch.

Zu guter Letzt möchte ich noch erwähnen, dass ich für einige Testnetze, unter anderem für Abb. 11, die Ergebnisse der Lastflussberechnung mit *PSS SINCAL* verfiziert habe um Fehler in der Modellierung der Netzelemente

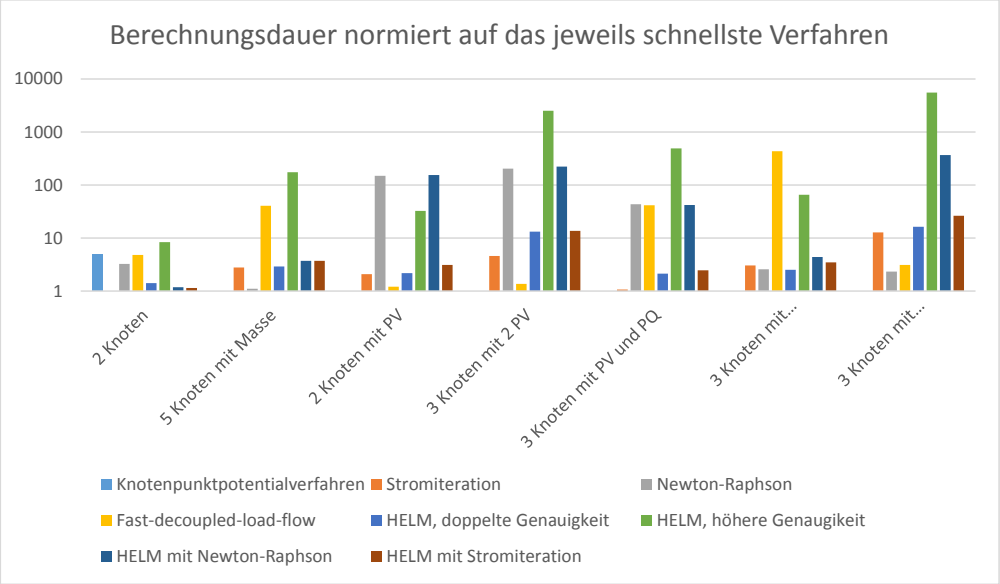

Abbildung 6: durchschnittliche Berechnungsdauer der Verfahren

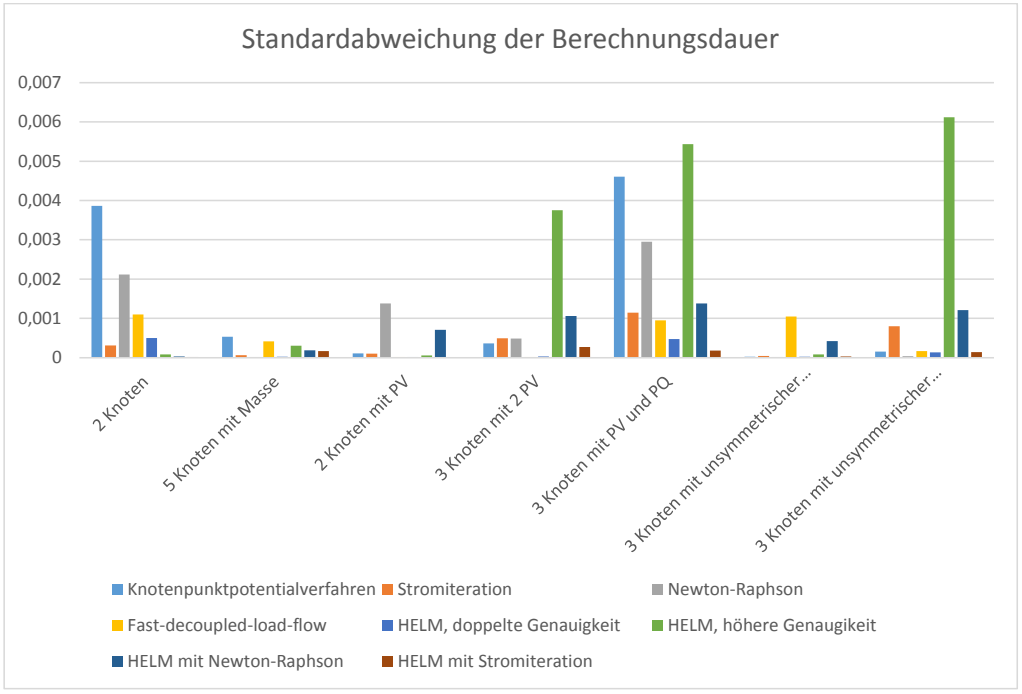

Abbildung 7: Standardabweichung der Berechnungsdauer der Verfahren

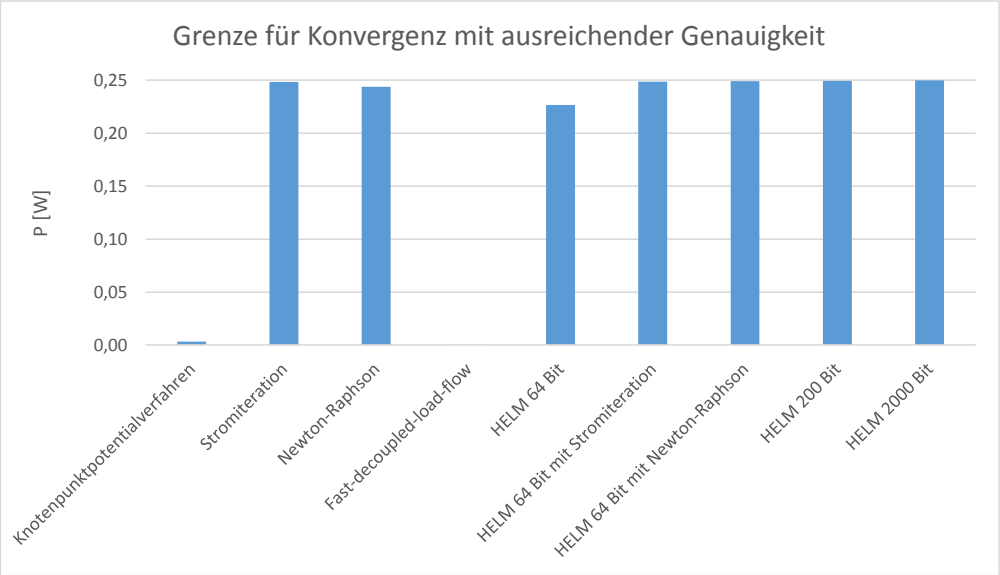

Abbildung 8: Iterativ ermittelte Lastgrenze, bei der das Verfahren noch mit einer ausreichenden Genauigkeit konvergiert

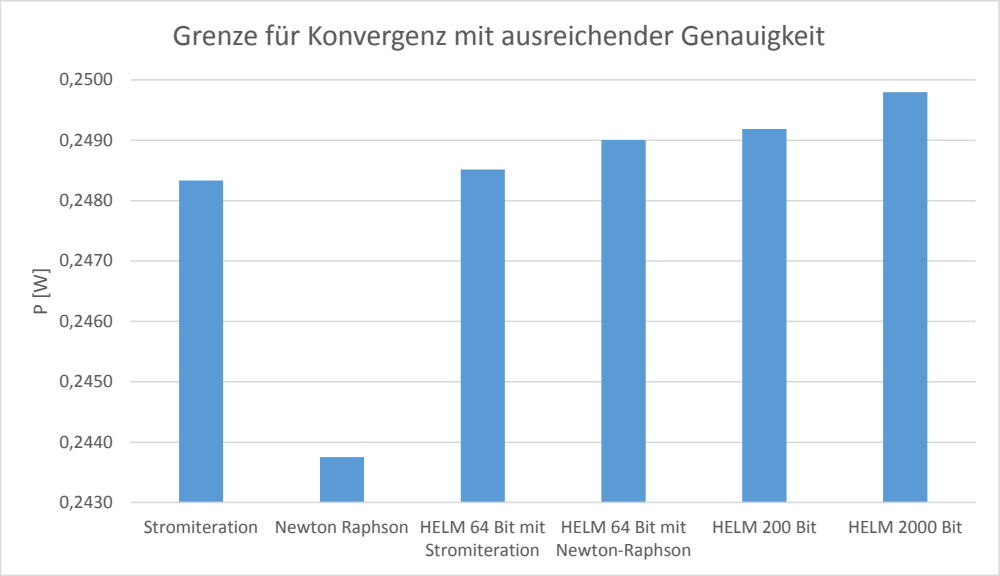

Abbildung 9: Iterativ ermittelte Lastgrenze, bei der das Verfahren noch mit einer ausreichenden Genauigkeit konvergiert, reduziert um das *Knotenpunktpotentialverfahren*, *FDLF* und *HELM* mit 64 Bit

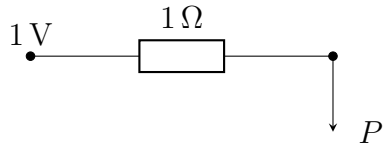

Abbildung 10: Netz zur Ermittlung der Konvergenzgrenze

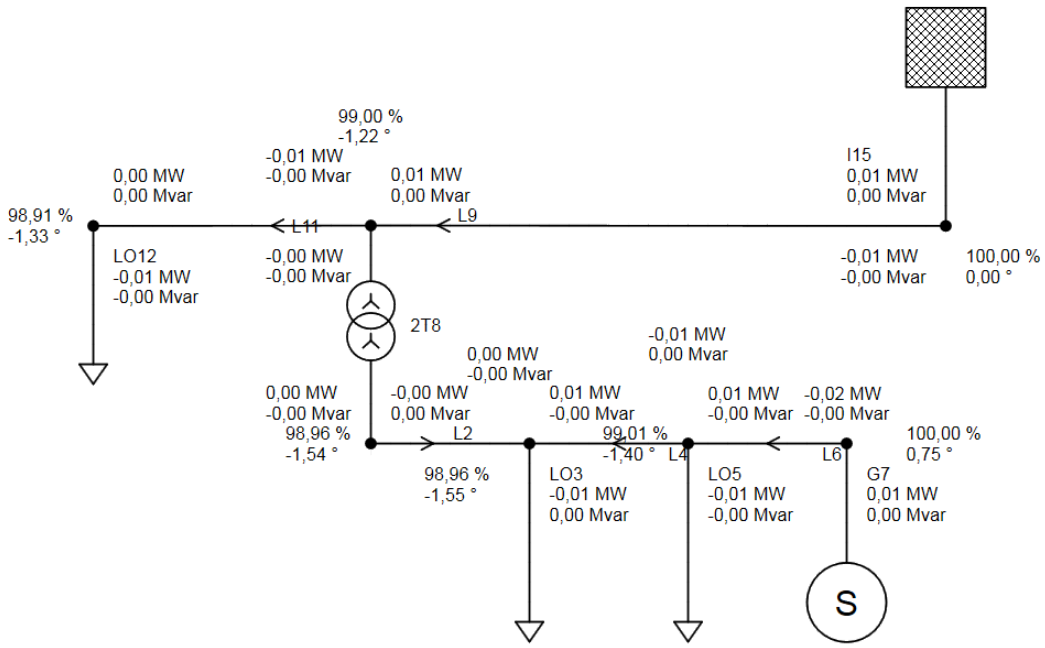

Abbildung 11: Testnetz zur Überprüfung der Modellierung

ausschließen zu können.

5 Fazit

HELM verspricht theoretisch die idealen Eigenschaften für ein Verfahren zur Lastflussberechnung. Aufgrund numerischer Einschränkungen, bedingt durch eine begrenzte Rechengenauigkeit, ergibt sich allerdings ein durchaus relevanter Nachteil: Man muss für *HELM*, in Reinform, abwägen zwischen Genauigkeit und Rechenzeit. Die Verschlechterung der Performance führt dabei zu Rechenzeiten welche um Größenordnungen schlechter sind als jene für die klassischen Verfahren wie die *Stromiteration* oder *Newton-Raphson*. Wenn man allerdings *HELM* mit nur 64 Bit mit einem iterativen Verfahren verbindet ist es möglich die Vorteile beider Verfahren zu kombinieren.

Schlussendlich ist also *HELM* kein Allheilmittel, es stellt allerdings eine durchaus nützliche Alternative dar. Welches Berechnungsverfahren nun das ideale ist hängt sehr stark von der jeweiligen Situation ab. Mit *HELM* hat man dabei allerdings eine weitere Option, welche sich insbesonders für Systeme nahe am Spannungszusammenbruch anbietet.

Literatur

[1] A. Trias, *The Holomorphic Embedding Load Flow Method*, IEEE PES General Meeting, July 2012 1, 2

[2] M. K. Subramanian, Y. Feng, D. Tylavsky, *PV Bus Modeling in a Holomorphically Embedded Power-Flow Formulation*, 978-1-4799-1255-1/13, IEEE, 2013 2

[3] A. Trias, *System and Method for Monitoring and Managing Electrical Power Transmission and Distribution Networks*, US Patent 7,519,506 B2, April 2009 2

[4] A. Trias, *System and Method for Monitoring and Managing Electrical Power Transmission and Distribution Networks*, US Patent US 2009/0228154 A1, September 2009 2, 2

[5] P. Wynn, *The Epsilon Algorithm and Operational Formulas of Numerical Analysis*, June 1960 2