Universität Bielefeld

Fakultät für Chemie

Masterarbeit

Eine graphische Benutzeroberfläche für hochdimensionale Quantendynamiksimulationen

Bearbeiter: Peter Protassow

Prüfer: Prof. Dr. Uwe Manthe

Zweitprüfer: Prof. Dr. Wolfgang Eisfeld

Abgabedatum: 18. Juni 2018

Hiermit versichere ich, die vorgelegte Masterarbeit selbstständig und ohne unzulässige Hilfe angefertigt zu haben. Die verwendeten Quellen und Hilfstexte sind vollständig angegeben und die Stellen der Arbeit, einschließlich Tabellen und Abbildungen, die anderen Werken im Sinn und Wortlaut entnommen wurden, als Entlehnung kenntlich gemacht. Die Bestimmungen der Masterprüfungsordnung sind mir bekannt. Die von mir vorgelegte Masterarbeit wurde in der Zeit vom 29. November 2017 bis 18. Juni 2018 im Arbeitskreis von Prof. Dr. Uwe Manthe an der Fakultät für Chemie der Universität Bielefeld unter der wissenschaftlichen Anleitung von Roman Ellerbrock durchgeführt.

Bielefeld, den 18. Juni 2018 .

. . . . . .

(Peter Protassow)

Inhaltsverzeichnis

1 Einleitung **1**

2 Theorie **3**

2.1 Ansatz der Multilayer-MCTDH-Wellenfunktion 5

3 Technische Beschreibung **11**

4 Ergebnisse **15**

4.1 Python-Interface für MCTDH 15

4.2 Graphische Benutzeroberfläche für MCTDH 22

5 Fazit und Ausblick **28**

Literaturverzeichnis **29**

1 Einleitung

Das zeitabhängige Multikonfiguration-Hartree-Verfahren (MCTDH)[1,2] und seine Multilayererweiterung (ml-MCTDH)[3,4] sind effiziente Verfahren zur genauen Simulation mehrdimensionaler Quantendynamik, die von mehreren Forschungsgruppen verwendet werden[5–23]. Beispiele für hochdimensionale Benchmark-Anwendungen sind die 21-dimensionalen Rechnungen, in denen die Tunnelaufspaltung des Grundzustands[24–29] und der angeregten[25–29] Schwingungszustände von Malonaldehyd erforscht wurden. Es wurden in 15-dimensionalen Rechnungen die Schwingungszustände von protonierten Wasserdimeren[30–34] untersucht. Außerdem wurde in 12-dimensionalen Rechnungen die thermischen Geschwindigkeitskonstanten[35–40], anfangszustandsausgewählte Reaktionswahrscheinlichkeiten[41–45] und die state-to-state Reaktionswahrscheinlichkeiten[46] für die Reaktion von Methan mit Wasserstoff untersucht. In diesen Rechnungen wurden detaillierte *ab initio* Potentialflächen verwendet. Signifikant höhere Dimensionen wurden in MCTDH-Rechungen mit Modelhamiltonoperatoren untersucht. So wurde in wegweisenden Rechnungen die nichtadibatischen Dynamiken von Pyrazin erforscht, in denen ein 24-modenschwingungsgekoppelter Hamiltonoperator[47–49] verwendet wurde. Multilayer-MCTDH Simulationen von typischen physikalischen Modellen[3,8,50–53] kondensierter Materie schließen üblich tausende Freiheitsgrade ein. Für die Untersuchung eines Photodissoziationsmodels wurden in einem Wirt-Gast-Komplex 189-dimensionale Multilayer-MCTDH Rechnungen durchgeführt[54]. Für weiterführende Literatur, in der das MCTDH-Verfahren und seine Anwendungen diskutiert werden, siehe Refs.[55–60].

Das MCTDH-Programmpaket, welches zur Berechnung und Simulation der oben genannten Systeme verwendet wird, bietet die Möglichkeit, Wellenfunktionen und Dichtematrizen hocheffizient zu propagieren, sowie Eigenzustände zu berechnen. Bisher ist die Bedienung des vorliegenden MCTDH-Programmpakets selbst bei Standardaufgaben Spezialisten vorenthalten und ein tiefgreifendes Verständnis der Programmstruktur ist erforderlich. Im Rahmen dieser Masterarbeit wurden zwei wesentliche Verbesserungen unternommen: Es wurde eine Benutzeroberfläche (von englisch graphical user inter-

face, GUI) erstellt, die Wissenschaftlern ohne Programmierkenntnisse die Bedienung des Programms bei Standardaufgaben ermöglicht. Des Weiteren wurde eine Python-Schnittstelle für MCTDH entwickelt, die es erlaubt, komplizierte Aufgaben mithilfe von Skripten zu bewältigen. Diese Schnittstelle ermöglicht die Nutzung des MCTDH-Programmpakets, ohne dass dessen Programmstruktur verstanden werden muss. Python hat sich als einsteigerfreundliche Programmiersprache erwiesen. Vergleichbare Python-Schnittstellen existieren auch für viele andere bekannte und häufig genutzte numerische Programmpakete(TensorFlow[61], SciPy[62] u.w.).

Diese Arbeit ist wie folgt gegliedert: In Kapitel 2 wird der Ansatz der MCTDH-Wellenfunktion beschrieben. Es werden die Unterschiede zum Multilayer-MCTDH herausgestellt. Die technische Beschreibung der GUI erfolgt in Kapitel 3. In Kapitel 4 werden die Python-Schnittstelle und die graphischen Benutzeroberfläche beschrieben. Schließlich wird in Kapitel 5 ein Fazit gezogen und ein Ausblick gegeben.

2 Theorie

Die Effizienz des MCTDH-Verfahrens resultiert aus der Doppellayerstruktur der verwendeten Wellenfunktion. In anderen Wellenpaktdynamikverfahren, wie der Standardmethode[60], wird die Wellenfunktion direkt in einer zeitunabhängigen Basis oder einem zeitunabhängigen Gitter dargestellt. Anstelle die Wellenfunktion in einer zeitunabhängigen Basis zu entwickeln, erfolgt im MCTDH-Verfahren die Darstellung des korrelierten mehrdimensionalen Wellenpaketes in einem Satz zeitabhängiger Basisfunktionen. Diese zeitabhängigen Basisfunktionen werden Einteilchenfunktionen (SPF) genannt. Die SPFs werden in der primitiven zeitunabhängigen Basis oder Gitter ausgedrückt. Das MCTDH-Verfahren kann als eine Zweilagendarstellung angesehen werden: So bilden die Entwicklungskoeffizienten, die genutzt werden, um die korrelierte Wellenfunktion in dem Satz der SPF-Basis auszudrücken, die obere Lage. Die zeitabhängigen Entwicklungskoeffizienten, die die zeitabhängigen SPFs in der primitiven zeitunabhängigen Basis oder Gitter darstellt, bilden die untere Lage. Die Bewegungsgleichungen, durch die gleichzeitig die optimale Entwicklungskoeffizienten für beide Lagen bestimmt werden, ergeben sich aus dem Dirac-Frenkel Variationsprinzip.

Dennoch ist auch das MCTDH durch die Anzahl der korrelierten Koordinaten limitiert. Die Effizienz des MCTDH-Verfahrens resultiert aus der Größe der SPF-Basis, welche, verglichen mit der primitiven Basis, signifikant kleiner gewählt werden kann. Allerdings skaliert der numerische Aufwand des MCTDH-Verfahrens exponentiell mit der Anzahl der korrelierten Koordinaten. Um Korrelationseffekte beschreiben zu können, sind mindestens zwei SPFs in jeder dieser Koordinaten notwendig. Die Anzahl der Konfigurationen, die in der MCTDH-Wellenfunktion enthalten ist, beträgt bei f korrelierten Koordinaten 2^f Konfigurationen. Aufgrund dieser Limitierung konnten mit dem MCTDH-Verfahren Systeme mit maximal 12-14 korrelierten Koordinaten berechnet werden. [35–37,47,63–66]

Im von Meyer eingeführten Moden-Kombinationsverfahren kann die Anzahl der Konfigurationen reduziert werden. In diesem Verfahren werden die „logischen“ Koordina-

ten, die in der MCTDH-Darstellung verwendet werden, von physikalischen Koordinaten unterschieden. Es werden verschiedene physikalische Koordinaten zu einzelnen logischen Koordinaten kombiniert. Analog zur Theorie der Elektronenstruktur werden diese mehrdimensionalen logischen Koordinaten Partikel genannt. Folglich wird die MCTDH-Rechnung statt durch die korrelierten Koordinaten durch die Partikel limitiert. So konnten Systeme mit 15-24 korrelierten Freiheitsgraden [31,32,49,67] und System-Bad-Modelle [68–70] durch das Modenkombinationsverfahren behandelt werden. Zwar konnte durch dieses Verfahren die Grenze zu höherer Dimensionalität verschoben werden, dennoch bleibt die grundlegende Einschränkung: Der numerische Aufwand skaliert mindestens 2^p, wobei p die Anzahl der logischen Koordinaten bzw. Partikel wiedergibt. Die Anzahl an physikalischen Koordinaten, die zu logische Koordinaten zusammengefasst werden können, ist begrenzt, da die SPFs nun mehrdimensionale Wellenfunktionen darstellen. Für molekulare Systeme stellte sich heraus, dass die Kombination von mehr als drei bis vier Koordinaten in einem Partikel ineffizient ist.

Die Begrenzung durch die Anzahl der korrelierten Koordinaten bzw. Partikel konnte durch das Multilayer-MCTDH-Verfahren [3] überwunden werden. Die SPFs des MCTDH-Verfahrens können ebenfalls als MCTDH-Wellenfunktion dargestellt werden. So kann z.B. eine MCTDH-Wellenfunktion um eine Lage erweitert werden, sodass die Wellenfunktion in drei Lagen ausgedrückt wird: Die oberste Lage wird durch die zeitabhängigen Entwicklungskoeffizienten gebildet. Die Wellenfunktion wird in der SPF-Basis der ersten Lage dargestellt, d.h. den SPFs des einfachen MCTDHs. Die mittlere Lage wird durch die zeitabhängigen Entwicklungskoeffizienten gebildet, die die SPFs der ersten Lage in der SPF-Basis der zweiten Lage darstellen. D.h. die SPFs der ersten Lage werden in der SPF-Basis einer zusätzlichen Lage ausgedrückt, die im Multilayer-MCTDH-Verfahren hinzugekommen ist. Schließlich werden in der untersten Lage die SPFs der zweiten Lage in der primitiven zeitunabhängigen Basis oder Gitter dargestellt. Durch eine rekursive Anwendung des MCTDH-Verfahrens können weitere Lagen hinzugefügt werden. Mit dem Multilayer-MCTDH-Verfahren waren quantumdynamische Rechnungen von System-Bad Modellen mit bis zu 1000 korrelierten Koordinaten möglich, in denen Elektronentransferprozesse [3,50] untersucht wurden.

Die Propagation der MCTDH-Wellenfunktion setzt die effiziente Berechnung der Matrixelemente des Hamiltonoperators voraus. Solange der Hamiltonoperator der Summe von Produkten von Einteilchenoperatoren [2] entspricht, stellt die Berechnung der Matrixelemente kein Problem dar. Im Gegensatz zu vielen Modelhamiltonoperatoren können *ab*

initio Potentialenergieflächen aber nur selten in dieser Form dargestellt werden. Durch die Verwendung einer spezifischen zeitabhängigen Quadratur, welche die Matrixelemente allgemeiner Potentiale effizient auswertet, können auch Matrixelemente resultierend aus *ab initio* Potentialenergieflächen effizient berechnet werden. Dieses Vorfahren wird correlation discrete variable representation (CDVR)[71,72] genannt.

Das ursprüngliche Vorgehen für das CDVR[71] beruht auf einem zeitabhängigen DVR-Gitter, das einer SPF-Basis entspricht. Somit kann das Standard-CDVR weder für modenkombinierte MCTDH-Rechnungen noch für Berechnungen mit dem Multilayer-MCTDH-Ansatz verwendet werden.

Hingegen ist das mehrdimensionales CDVR-Verfahren mit dem modenkombinierten MCTDH-Verfahren kompatibel.[73] Der numerische Aufwand des CDVRs hängt linear von der Anzahl der verwendeten primitiven Gitterpunkten ab, die für die Darstellung der SPFs benötigt werden. In modenkombinierte MCTDH-Rechnungen wird eine große Anzahl an primitiven Gitterpunkten verwendet, sodass modenkombinierte MCTDH-Rechnungen kombiniert mit der CDVR- Auswertung des Potentials ineffizient sind. Multilayer-MCTDH-Rechnungen benötigen dagegen keine mehrdimensionalen Gitter, um die SPFs darzustellen und bieten sich daher in Kombination mit dem CDVR an.

2.1 Ansatz der Multilayer-MCTDH-Wellenfunktion

Zunächst werden die Wellenfunktionen der Standardmethode, des Zweilagen-MCTDHs und des modenkombinierten MCTDH betrachtet, um anschließend den Ansatz der Multilayer-MCTDH-Wellenfunktion stufenweise vorzustellen.

In der Standardmethode wird die Wellenfunktion in einer zeitunabhängigen Basis bzw. zeitunabhängigen Gitter dargestellt. Die mehrdimensionale Wellenfunktion wird durch das direkte Produkt von eindimensionalen Basisfunktionen $\phi_j^{\kappa}(x_\kappa)$ wie folgt dargestellt:

$$\Psi(x_1, ..., x_f, t) = \sum_{j_1=1}^{N_1} ... \sum_{j_f=1}^{N_f} A^1_{j_1,...,j_f}(t) \cdot \mathcal{X}^{(1)}_{j_1}(x_1) \cdot ... \cdot \mathcal{X}^{(f)}_{j_f}(x_f) \tag{2.1}$$

Die zeitabhängigen Koeffizienten $A^1_{j_1,...,j_f}(t)$ beschreiben die Bewegung der Wellenpakete. Die Darstellung der Wellenfunktion in Gleichung 2.1 kann auch als einlagige Darstellung angesehen werden und die Hochzahl 1 von $A^1_{j_1,...,j_f}(t)$ soll darauf hinweisen, dass $A^1_{j_1,...,j_f}(t)$ ein Entwicklungskoeffizient der ersten (und einzigen) Lage ist.

Im MCTDH-Verfahren wird eine zusätzliche Lage für die Darstellung der Wellenfunktion eingeführt. Die mehrdimensionale Wellenfunktion wird erst in einer orthonormalen Basis der zeitabhängigen SPFs $\phi_j^{1;\kappa}(x_\kappa, t)$ entwickelt.

$$\Psi(x_1, ..., x_f, t) = \sum_{j_1=1}^{n_1} ... \sum_{j_f=1}^{n_f} A^1_{j_1,...,j_f}(t) \cdot \phi_{j_1}^{1;1}(x_1, t) \cdot ... \cdot \phi_{j_f}^{1;f}(x_f, t). \tag{2.2}$$

Anschließend werden diese SPFs innerhalb der zeitunabhängigen primitive Basis dargestellt:

$$\phi_m^{1;\kappa}(x_\kappa, t) = \sum_{j=1}^{N_\kappa} A^{2;\kappa}_{m;j}(t) \cdot \mathcal{X}_j^{(\kappa)}(x_1). \tag{2.3}$$

Gleichung 2.3 beinhaltet einen Satz zusätzlicher Entwicklungskoeffizienten, $A^{2;\kappa}_{m;j}(t)$, der die zeitabhängigen SPFs in der zeitunabhänigen primitiven Basis darstellt. Die hochgestellte Zahl z der Koeffizienten $A^z(t)$ bezieht sich auf die Lagentiefen. In Gleichung 2.3 folgt aus $z = 2$, das Gleichung 2.3 die zweite Lage darstellt. Das hochgestellte κ und der Index m von $A^{2;\kappa}_{m;j}(t)$ beziehen sich auf die m-te SPF der κ-te Koordinate.

Zur Visualisierung unterschiedlicher MCTDH-Wellenfunktionen, für die unterschiedlich viele Lagen verwendet wurden, dienen Baumdiagramme, die in Abbildung 2.1.1 abgebildet sind. Als Beispiel soll ein siebendimensionales System dienen. Die Standardwellenpaketdarstellung aus Gleichung 2.1 und die MCTDH-Darstellung sind in Abbildung 2.1.1a und 2.1.1b schematisch dargestellt. In den Diagrammen sind die verschiedenen Sätze der A-Koeffizienten durch die ausgefüllten schwarzen Kreise gekennzeichnet. So kommt in Abbildung 2.1.1a ein Satz von Koeffizienten $A^1_{j_1,...,j_7}$ vor, der durch den einzigen schwarzen Punkt gekennzeichnet ist. Jede Linie, die von solchen Kreisen führt, entspricht einem tiefgestellten Index aus $A^{2;1}_{m;j}$ und die Zahl neben einer Linie gibt den maximalen Wert des Indexes an, d.h. die jeweilige Basisgröße. Die tieferliegende primitive Darstellung wird durch den Koordinatdeskriptor x_n hervorgehoben. Beispielsweise ist der Koeffizienten $A^{2;1}_{m;j}$, der durch die SPFs $\phi_m^{1;1}(x_1, t)$ dargestellt wird, mit dem Koeffizienten A^1 über eine Linie verbunden, die den m-ten Index darstellt. Über eine weitere Linie, die für den j-ten Index steht, ist $A^{2;1}_{m;j}$ mit dem Koordinatdeskriptor x_1 verbunden. In Abbildung 2.1.1a ist der Koeffizient $A^1_{j_1,...,j_7}$ durch sieben Linien mit den entsprechenden Indizes von j_1 bis j_7 direkt mit den sieben Koordinatdeskriptoren $x_1, x_2, ..., x_7$ verbunden und in Abbildung 2.1.1b. ist $A^1_{j_1,...,j_7}$ mit den sieben Sätzen an A-Koeffizienten $A^{2;1}, A^{2;2}, ..., A^{2;7}$ verbunden. Die Wellenfunktion aus Abbildung 2.1.1a ist direkt in der zeitunabhängi-

gen primitiven Basis dargestellt. In Abbildung 2.1.1b gibt es eine intermediäre Lage an zeitabhängigen SPFs.

Während in den Gleichungen 2.2 und 2.3 nur eindimensionale SPFs vorkommen, werden im modenkombinierten MCTDH-Verfahren mehrdimensionale SPFs verwendet. Hierfür werden die f physikalischen Koordinaten $x_1, x_2, ..., x_f$ d logischen Gruppen zugeordnet, die Partikel genannt werden. Die mehrdimensionalen Koordinaten $q_1^1, q_2^1, ..., q_d^1$ sind wie folgt definiert:

$$\begin{aligned} q_1^1 &= \{q_1^{2;1}, q_2^{2;1}, q_{d_1}^{2;1}\} \\ &= \{x_1, x_2, ..., x_{d_1}\} \\ q_2^1 &= \{q_1^{2;2}, q_2^{2;2}, q_{d_2}^{2;2}\} \\ &= \{x_{d_1+1}, x_{d_1+2}, ..., x_{d_1+d_2}\} \\ &\vdots \\ q_{f^1}^1 &= \{q_1^{2;d}, q_2^{2;d}, q_{d_d}^{2;d}\} \\ &= \{..., x_f\} \end{aligned}$$

Die logische mehrdimensionale Koordinate q_κ^1 umfasst d_κ Koordinaten: $q_1^{2;\kappa}, q_2^{2;\kappa}, ..., q_{d_\kappa}^{2;\kappa}$. Die Hochzahl 1 und 2 in dieser Notation zeigt, ob die Koordinate eine mehrdimensionale Koordinate der ersten Lage ist oder einer Koordinate der zweiten Lage entspricht. Für die Koordinate der zweiten Lage gibt der zusätzliche hochgestellter Index κ den Index der Koordinate der ersten Lage an. Eine modenkombinierte MCTDH-Wellenfunktion mit d logischen Koordinaten wird wie folgt definiert:

$$\Psi(q_1^1, q_2^1, ..., q_d^1, t) = \sum_{j_1=1}^{n_1} ... \sum_{j_d=1}^{n_d} A^1_{j_1,...,j_d}(t) \cdot \phi^{1;1}_{j_1}(q_1^1, t) \cdot ... \cdot \phi^{1;d}_{j_d}(q_d^1, t) \tag{2.4}$$

$$\begin{gathered} \phi_m^{1;\kappa}(q_\kappa^1, t) = \sum_{j=1}^{N_\alpha} ... \sum_{j_{d_\kappa}=1}^{N_\beta} A^{2;\kappa}_{m;j_1,...,j_{d_\kappa}}(t) \cdot \mathcal{X}^{(\alpha)}_{j_1}(q_1^{2;\kappa}) \cdot ... \cdot \mathcal{X}^{(\beta)}_{j_{d_\kappa}}(q_{d_\kappa}^{2;\kappa}) \\ \left(\alpha = 1 + \sum_{i=1}^{\kappa-1} d_i \text{ and } \beta = 1 + \sum_{i=1}^{\kappa} d_i\right) \end{gathered} \tag{2.5}$$

In Abbildung 2.1.1c ist das entsprechende Diagramm der Wellenfunktion eines siebendimensionalen Systems dargestellt, dessen Koordinaten in logischen Koordinaten zusammengefasst wurden. Die physikalischen Koordinaten bilden in Abbildung 2.1.1c drei mehrdimensionale logische Koordinaten: $q_1^1 = \{x_1, x_2\}$, $q_2^1 = \{x_3, x_4\}$ und $q_3^1 = \{x_5, x_6, x_7\}$.

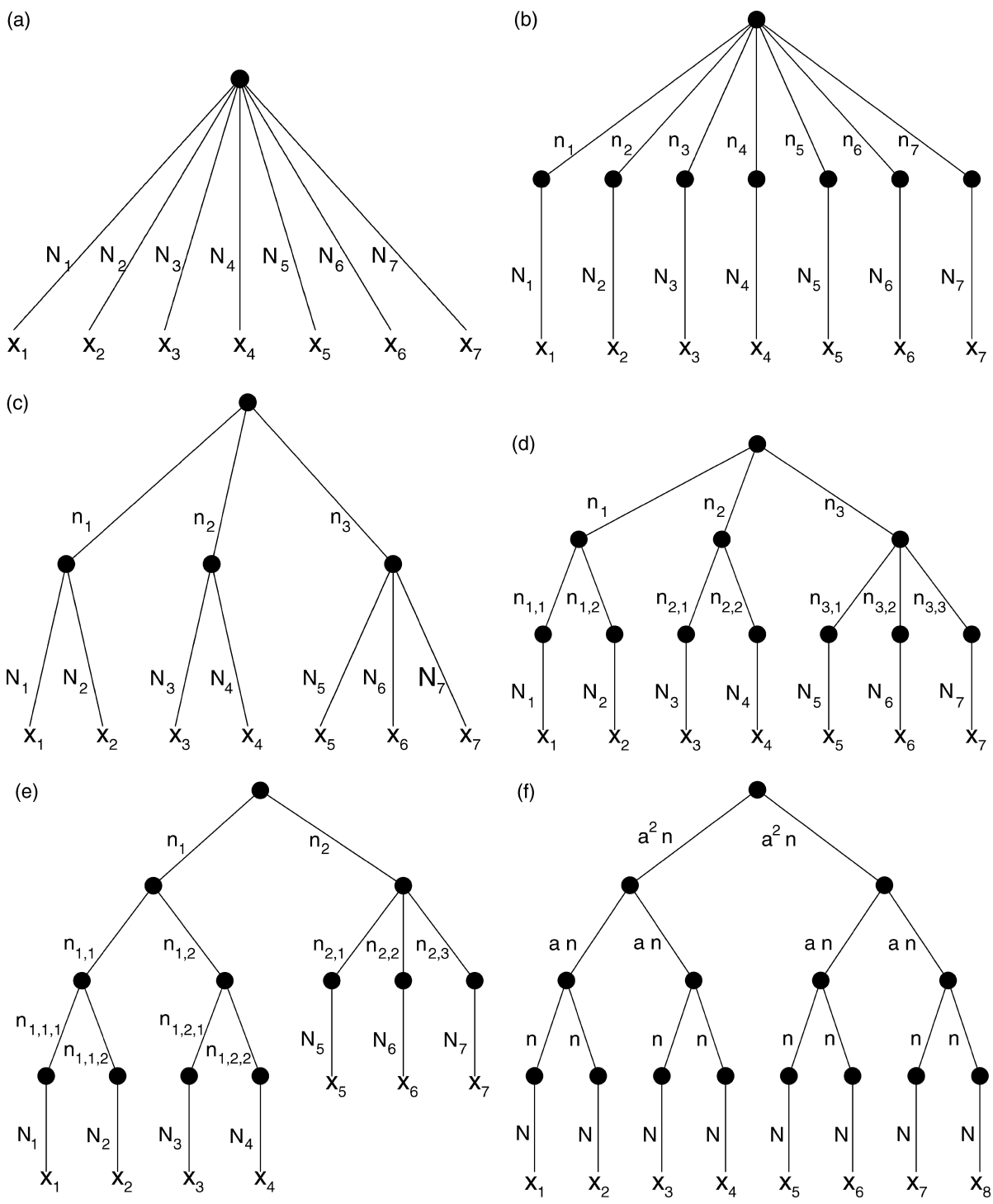

Abbildung 2.1.1: Unterschiedliche Darstellungen von Wellenfunktionen eines siebendimensionalen Systems. Dargestellt sind: (a) Eine Darstellung eines Standardwellenpakets, (b) eine MCTDH-Wellenfunktion, (c) eine modenkombinierte MCTDH-Wellenfunktion, [(d)-(f)] eine Multilayer-MCTDH-Wellenfunktion. Die Diagramme wurden aus Ref.[4] entnommen.

Im einfachsten Fall kommen im Multilayer-MCTDH zwei Lagen von SPFs vor. Anstelle die SPFs aus Gleichung 2.4 in der primitiven zeitunabhängigen Basis zu entwickeln, wie es bereits im modenkombinierte MCTDH-Verfahrens in Gleichung 2.4 durchgeführt wurde, können die mehrdimensionalen SPFs ebenfalls durch das MCTDH-Verfahren dargestellt werden. Diese Entwicklung resultiert in einer Multilayer-MCTDH-Wellenfunktion. Der Ansatz der Multilayer-MCTDH-Wellenfunktion ist gegeben durch

$$\Psi(q_1^1, q_2^1, ..., q_d^1, t) = \sum_{j_1=1}^{n_1} ... \sum_{j_d=1}^{n_d} A^1_{j_1,...,j_d}(t) \cdot \phi^{1;1}_{j_1}(q_1^1, t) \cdot ... \cdot \phi^{1;d}_{j_d}(q_d^1, t) \tag{2.6}$$

$$\begin{aligned} \phi^{1;\kappa}_m(q^1_\kappa, t) &= \phi^{1;\kappa}_m(q_1^{2;\kappa} ..., q_{d_\kappa}^{2;\kappa}, t) \\ &= \sum_{j=1}^{n_{\kappa,1}} ... \sum_{j_{d_\kappa}=1}^{n_{\kappa,d_\kappa}} A^{2;\kappa}_{m;j_1,...,j_{d_\kappa}}(t) \\ &\quad \times \phi^{2;\kappa,1}_{j_1}(q_1^{2;\kappa}, t) \cdot ... \cdot \phi^{2;\kappa,d_\kappa}_{j_{d_\kappa}}(q_{d_\kappa}^{2;\kappa}, t), \\ \phi^{2;\kappa,\lambda}_m(q^{2;\kappa}_\lambda, t) &= \sum_{j=1}^{N_\alpha} A^{3;\kappa,\lambda}_{m;j}(t) \mathcal{X}^{(\alpha)}_j(q^{2;\kappa}_\lambda) \end{aligned} \tag{2.7}$$

$$\left(\text{mit } \alpha = \lambda + \sum_{i=1}^{\kappa-1} d_i \right). \tag{2.8}$$

In Gleichung 2.7 ist $\phi^{2;\kappa,\lambda}_m$ mit der dazugehörigen Koordinate $q^{2;\kappa}_\lambda$ eine SPF der zweiten Lage. Die Hochzahl 2 bezieht sich auf die Lagentiefe, sodass die SPF zur zweiten Lage gehört, wobei κ und λ die dazugehörigen Koordinaten kennzeichnen. Die Entwicklungskoeffizienten $A^{3;\kappa,\lambda}_{m;j}(t)$ werden verwendet, um diese SPF darzustellen und die Entwicklungskoeffizienten $A^{2;\kappa}_{m;j_1,...,j_{d_\kappa}}(t)$ definieren die Entwicklung der SPFs der ersten Lage, die in der SPF-Basis der zweiten Lage entwickelt werden. In Gleichung 2.3 definierten diese Entwicklungskoeffizienten die SPFs der ersten Lage in der primitiven zeitunabhängigen Basis. Abbildung 2.1.1d zeigt das entsprechende Diagramm für ein siebendimensionales System in Multilayer-MCTDH-Darstellung.

Da die Gleichungen der Multilayer-MCTDH-Wellenfunktion schnell unhandlich werden, ist es einfacher, statt der Gleichungen die Wellenfunktionen wie in Abbildung 2.1.1 als Diagramm anzugeben. Jedes Diagramm definert die Wellenfunktionen eindeutig, sodass aus den Diagrammen die entsprechenden Gleichunge der Wellenfunktionen abgeleitet werden können. Die Notation für die SPFs, A-Koeffizienten und (mehrdimensionalen) Koordinaten, die oben in den Gleichungen angegeben wurde, kann einfach für beliebig

mehrlagige Darstellungen erweitert werden. Ein Beispiel dieser Strukturen ist in Abbildung 2.1.1e dargestellt. Die Anzahl der Lagen für die jeweiligen Koordinaten kann bis zur primitiven Darstellung von Koordinate zu Koordinate variieren. So wird in Abbildung 2.1.1e das Baumdiagramm einer Multilayer-MCTDH-Wellfunktion gezeigt, in der für die Koordinaten $x_1 - x_4$ drei MCTDH-Lagen verwendet werden (d.h. insgesamt eine Vier-Lagen-Darstellung) und für die Koordinaten $x_5 - x_7$ werden zwei MCTDH-Lagen verwendet (d.h. eine Drei-Lagen-Darstellung).

3 Technische Beschreibung

Die graphisch Benutzeroberfläche (GUI) für MCTDH-Rechnungen wurde in Python und Qt implementiert. Der Zugriff auf die Qt-Bibliothek erfolgt über die Python-Bibliothek PyQt4. PyQt4 umfasst zehn Python-Module, die zusammen ungefähr 400 Klassen und 6000 Methoden und Funktionen enthalten.[74]

In Abbildung 3.0.1 sind die wichtigen Klasse aufgeführt, die für die Implementierung der GUI verwendet wurden. Die Klassen, die in Rechtecken zusammengefasst wurden, entstammen aus Python-Modulen, deren Namen links über den Rechtecken angegeben sind. Bei den Modulen handelt es sich um die PyQt4-Module *QtCore* und *QtGui*. Für die graphisch Darstellung der MCTDH-Baumdiagramme wurden die Module *matplotlib* und *networkx* verwendet. Die Klassen, die in Abschnitt 4.1 vorgestellt werden, sind im Modul *mctdh* enthalten.

Alle *mctdh*-Klassen werden in ModelTree verwendet, um alle Informationen zum MCTDH-Baum zu erhalten. Die Information wird an die Klasse *LogicalNodes* übergeben und es werden in dem Datentyp *Dictionary* die Knoten des Baums mit den SPFs und für den untersten Layers mit den Moden gespeichert. Speziell für die Visualisierung von Baumdiagramme existiert das Python-Modul *networkx*, aus dem die Klasse *diGraph* verwendet wird, in der die Knoten gespeichert werden. Das Baumdiagramm wird in der Klasse *View* in einem *png*-File mithilfe des Moduls *matplotlib* gespeichert, das für die Erzeugung von Diagrammen entwickelt wurde. Das *png*-File wird anschließend in der GUI verwendet. Die Informationen über den MCTDH-Baum wird von *LogicalNodes* auch an die Klasse *Tree* übergeben.

Die Pfeile mit den ausgefüllten Pfeilköpfen führen von Klassen, die in anderen Klassen verwendet werden, auf die die Pfeilspitze zeigt. Auf die Klassen, die durch Vererbung erstellt wurden, zeigen rot umrandete Pfeilspitzen. Beispielsweiße führen diese Pfeile von allen angegeben PyQt-Klassen , von denen geerbt wird. Sowohl von *QDialog* als auch *QMainWindow* werden durch Vererbung Unterklassen generiert: *dialogA, dialogc* und *Main*. Allerdings wurden diese drei Klassen in Qt-Designer erzeugt, in dem die jeweiligen

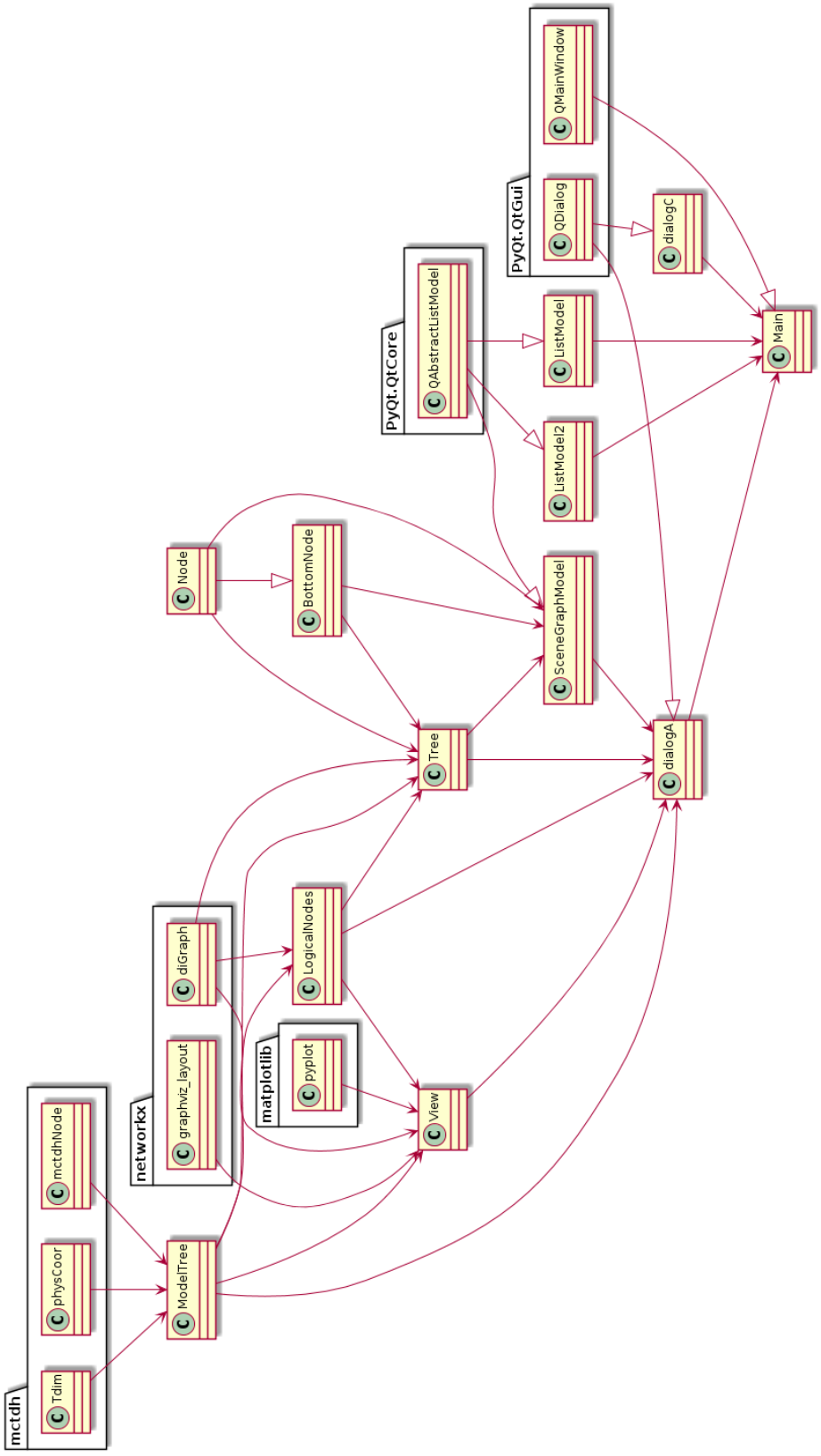

Abbildung 3.0.1: Klassendiagram der MCTDH-GUI. Eine Beschreibung des Diagramms findet sich im Text wieder.

todo: 10. Juni 2018 - 18:04 Uhr

Fenster mit den benötigten Steuerungselementen zusammengestellt werden können. So können die Größen der Steuerungselemente ohne Programmierung per Maus festgelegt werden. Die Informationen über die jeweiligen Fenster werden in *ui*-Dateien gespeichert. Mit PyQt können diese Dateien eingelesen werden und aus den Daten die entsprechenden Klassen erstellt und beliebig erweitert werden. Die beiden Klassen *QDialog* und *QMainWindow* stammen von *QWidget* ab. *QWidget*, *QDialog* und *QMainWindow* sind Steuerungselement, mit denen der Benutzer durch die Tastatur und Maus interagieren kann.[74]

Die Klasse *Main* stellt das Hauptfenster der GUI dar, von dem aus neue Projektordner erstellt, umbenannt oder gelöscht werden können. In diesen Ordner finden sich wiederum Ordner, die Einstellungen unterschiedlicher Rechnungen enthalten. Schließlich können aus dem Hauptfenster neben der Ordernderverwaltung MCTDH-Rechnungen gestartet werden. Die Klasse *dialogC* generiert eine Fenster, in dem die Ordnernamen eingetragen werden können, um entweder neue Ordner zu erstellen oder alte Ordner um zu benennen. Die Einstellungsparameter einer MCTDH-Rechnung werden in der Klasse *dialogA* angegeben. Bereits existierende MCTDH-Basisfiles werden eingelesen und im *dialogA*-Fenster dargestellt.

Alle Steuerungselement wie Knöpfe, Checkboxen oder Elemente innerhalb eines Fensters emittieren Signale aus, die Aktionen des Benutzers zugeordnet werden können. Aktionen können das Einmal- oder Doppeltklicken, das Bewegen des Mauszeigers oder das Betättigen der Entertaste sein. Einzelne Steuerungselemente können zusammen mit einer bestimmten Aktion mit einer Klassenmethode bzw. Funktion verbunden werden, die die Klassenmethoden auslösen.

Qt enthält Klassen, mit denen beliebig viele Elemente dargestellt werden können. Diesen Klassen liegt eine Model/View-Aufbau zugrunde, der das Datenmodel von der Darstellung der Daten trennt. Ein Datenmodel ist die Klasse *QAbstractListModel*, in die die Daten eingelesen, bearbeitet und gelöscht werden können. Die Daten können wiederum in den Klassen *QListView* und *QTreeView* dargestellt werden. Die Trennung zwischen dem Datenmodel und der graphischen Darstellung der Daten beruht auf dem Model-View-Controller (MVC) Paradigma.[75]

Bei der MVC-Programmierung werden verschiedener Klassen erstellt. Jede dieser Klassen erfüllt unterschiedliche Aufgaben: die Verarbeitung von Daten innerhalb der Anwendungssoftware (Model), die Visualisierung des aktuellen Systemzustandes (View) und die Interaktion zwischen Benutzer und Programm (Controller).[76]

In Qt wurden der Controller und View kombiniert, sodass die Speicherung und Bearbeitung der Daten von der Datenvisualisierung getrennt wurde. Die gleichen Daten können in verschieden Ansichten dargestellt werden. Die Implementierung neuer Darstellungsarten ändert nicht die darunterliegende Datenstruktur.[75] Der Vorteil der Model/View-Architektur ist, dass die Element, die die visualisierten Daten des Models darstellen, nicht jeweils mit einer Funktion gekoppelt werden muss wie bei anderen Steuerungselementen. So können Aktionen beliebig vieler Elemente mit nur einer Funktion verbunden werden. Dabei wird nur das Steuerungselement, das die Daten darstellt, mit den gewünschten Aktionen verbunden, wobei Aktionen auf ein beliebiges Element innerhalb der Steuerungselemente Informationen über dieses Element in Bezug auf das Datenmodel an die Funktion überträgt.

Qt besitzt für die Model/View-Architektur Standardmodel, allerdings können die Modelle durch die Vererbung von QAbstractListModel verändert und angepasst werden. So bekommt *SceneGraphModel* keine Liste als Eingabetype wie die Klassen *ListModel* und *ListModel2*, sondern Objekte der Klassen *Node* und *BottomNode*. Die Klasse *BottomNode* erbt von *Node* und enthält zusätzlich Informationen zu den Moden der untersten Knoten. *Node*-Objekte spiegeln bestimmte Knoten des MCTDH-Baums wieder, in denen Informationen zu Elternknoten und Kinderknoten gespeichert sind. Diese Objekte werden in der Klasse *Tree* in einem *Dictionary* zum Baum zusammengefasst. Die Daten des Models aus *SceneGraphModel* werden über die PyQt-Klasse QTreeView in *dialogA* visualisiert. *ListModel* und *ListModel2* enthält eine Liste der Projektordner und der Ordner verschiedener Rechnungen innerhalb der Projekte. Diese Daten werden in zwei getrennten QListView dargestellt und können mithilfe des Models aktualisiert werden.

4 Ergebnisse

4.1 Python-Interface für MCTDH

Es wurde eine Programmierschnittstelle (von englisch *application programming interface*, API) für das MCTDH-Programmpaket erstellt. Im Rahmen dieser Arbeit wurde sich auf Klassen beschränkt, welche für das Einlesen der baumförmig strukturierten MCTDH-Basis zuständig sind. Die Klassen und Methoden, die über Python aufgerufen werden können, sind in Abbildung 4.1.1 dargestellt. Jede Klasse der API wird in Abbildung 4.1.1 durch einen Kasten repräsentiert. Im oberen Teil des Kastens ist der Klassenname angegeben und im unteren Teil sind die Methoden der Klasse aufgelistet. Die Anordnung der Kästen gibt die Abhängigkeit der Klassen zueinander wieder. So muss ein Objekte von der Klasse *ControlParameter* generiert werden, um Objekte der Klasse *mctdhBasis* erzeugen zu können. Analog hängen die Klassen *Tdim* und *physCoor* von der Klasse *mctdhNode* ab. *ControlParameter* und *mctdhBasis* sind die Klassen, die die Konfigurations- und Basisdatei einlesen und die Genauigkeitsparameter und die MCTDH-Basis verwalten. Dagegen wird in der Klasse *mctdhNode* die lokale Information eines Knotens verwaltet. So können Knoteneigenschaften über die Klasse *mctdhNode* ermittelt werden. Die Knotenobjekte können in Nachbarknoten überführt werden. Die SPFs eines Knoten wird in der Klasse *Tdim* ermittelt. Mit der Klassen *physCoor* können die Schwingungsmoden, der untersten Knoten bestimmt werden.

Zur Demostration der API wird im folgenden ein Python-Skript vorgestellt, mithilfe dessen die Größe einer MCTDH-Wellenfunktion berechnet wird:

```
import mctdh

config = mctdh.controlParameters()
config.initialize('mctdh.config')
basis = mctdh.MctdhBasis()
basis.initialize('basis.txt', config)

maxNodes = basis.NmctdhNodes()

nodes_spf = {}
ProdBottomNode = {}
remnantNodeList = []

def get_SPFs():
    TopNode = 0
    remnant = 0

    for i in range(maxNodes):
        node = basis.MCTDHnode(i)
        tdim = node.t_dim()
        nodes_spf[i] = tdim.GetnTensor()

    primitivB = {i: basis.MCTDHnode(i).t_dim().active(0) for i in \
                range(maxNodes) if \
                basis.MCTDHnode(i).Bottomlayer() == True}

    for key in primitivB:
        ProdBottomNode[key] = primitivB[key] * nodes_spf[key]
    ProdBottom = sum([l_[1] for l_ in ProdBottomNode.items()])
```

```
        for i in range(maxNodes):
            if basis.MCTDHnode(i).Toplayer() == True:
                    children = basis.MCTDHnode(i).NChildren()
                    for j in range(children):
                        TopNode *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
                    TopNode *= basis.MCTDHnode(i).t_dim().GetnTensor()

        for i in range(maxNodes):
            if basis.MCTDHnode(i).Toplayer() == False and \
            basis.MCTDHnode(i).Bottomlayer() == False:
                    children = basis.MCTDHnode(i).NChildren()
                    parent = basis.MCTDHnode(i).t_dim().GetnTensor()
                    for j in range(children):
                        remnant *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
                    remnant *= parent
                    remnantNodeList.append(remnant)
                    remnant = 0
        remnantSum = sum(remnantNodeList)

        return ProdBottom + TopNode + remnantSum

    print get_SPFs()
```

Die in Abbildung 4.1.1 dargestellte Klassen können mit folgenden Befehl in Python aufgerufen werden:

```
import mctdh
```

Um in Python die MCTDH-Klassen verwenden zu können, genügt es *mctdh* dem Klassennamen voranzustellen und durch einen Punkt zu trennen.

```
config = mctdh.controlParameters()
basis = mctdh.MctdhBasis()
```

Über die initialisierten Objekte kann auf die Klassenmethoden zugegriffen werden:

```
config.initialize('mctdh.config')
basis.initialize('basis.txt', config)
```

Mithilfe des Objektes *basis* kann die Anzahl der Knoten des eingelesenen MCTDH-Baums bestimmt werden:

```
maxNodes = basis.NmctdhNodes()
```

Um die Größe der Wellenfunktion berechnen zukönnen, wird die Anzahl der SPFs eines Knotens und die Anzahl der SPFs aller Nachbarknoten multipliziert. Alle SPFs werden in den Dictionary-Datentyp *nodes_spf* gespeichert. Anschließend wird die Summe aller SPFs-Produkte gebildet. Hierfür werden die Summen der Produkte vom oberen Knoten, von den unteren Knoten und von denen restlichen Knoten separat gebildet und in einem Dictionary bzw. einer Liste gespeichert:

```
nodes_spf = {}
ProdBottomNode = {}
remnantNodeList = []
```

Der Datentyp Dictionary wird mit geschweiften Klammern und der Datentyp Liste wird mit eckigen Klammern deklariert. Die Berechnung der Größe der Wellenfunktion wird in einer Funktion vorgenommen:

```
def get_SPFs()
```

Die Anzahl der SPFs und primitiven Basis werden jeweils in verschiedene Dictionaries gespeichert:

```
for i in range(maxNodes):
    node = basis.MCTDHnode(i)
    tdim = node.t_dim()
    nodes_spf[i] = tdim.GetnTensor()

primitivB = {i: basis.MCTDHnode(i).t_dim().active(0) for i in \
            range(maxNodes) if \
            basis.MCTDHnode(i).Bottomlayer() == True}
```

In der for-Schleife werden alle Knoten des Baums erfasst und für jeden i-ten Knoten wird die Anzahl an SPFs in *nodes_spf* gespeichert. Die Anzahl der primitiven Basis wird in dem Dictionary *primitivB* gespeichert. Dabei wird über alle Knoten iteriert, die zu den untersten Knoten gehören und die Größe der primitiven Basis wird mit *active(0)* im Dictionary mit dem Index *i* gespeichert. Die Anzahl der SPFs und der primitiven Basis der untersten Knoten wird für jeden Knoten multipliziert. Anschließend werden die Produkte summiert und die Summe in textitProdBottom gespeichert:

```
for key in primitivB:
    ProdBottomNode[key] = primitivB[key] * nodes_spf[key]
ProdBottom = sum([l_[1] for l_ in ProdBottomNode.items()])
```

Die for-Schleife iteriert über alle Indizes des Dictionary *primitivB*. Da die untersten Knoten in *primitivB* dieselben Indizes besitzen, ist gewährleistet, dass nur Produkte aus der Anzahle der SPFs und der primitven Basis für jeden untersten Knoten berechnet werden.

Das Produkt aus SPF-Anzahl des ersten Knoten und seiner Nachbarknoten wird im folgenden Code-Abschnitt berechnet:

```
for i in range(maxNodes):
    if basis.MCTDHnode(i).Toplayer() == True:
            children = basis.MCTDHnode(i).NChildren()
            for j in range(children):
                TopNode *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
            TopNode *= basis.MCTDHnode(i).t_dim().GetnTensor()
```

Zunächst wird die Anzahl an Nachbarknoten berechnet und in der Variabel *children* gespeichert. Es wird über alle Nachbarn iteriert und das Produkt der SPF-Basisgröße aller Nachbarknoten und des obersten Knoten gebildet und in TopNode gespeichert.

Im letzten Code-Block wird die Summe aus den restlichen Produkten gebildet:

```
for i in range(maxNodes):
        if basis.MCTDHnode(i).Toplayer() == False and \
        basis.MCTDHnode(i).Bottomlayer() == False:
                children = basis.MCTDHnode(i).NChildren()
```

```
                parent = basis.MCTDHnode(i).t_dim().GetnTensor()
                for j in range(children):
                    remnant *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
                remnant *= parent
                remnantNodeList.append(remnant)
                remnant = 0
    remnantSum = sum(remnantNodeList)
```

Es wird über die restlichen Knoten iteriert, die Nachbarknoten werden bestimmt und die Produkte der Basisgrößen werden gebildet und in der Liste *remnantNodeList* gespeichert. Die Addition der Produkte aus den obersten, untersten und restlichen Knoten ergibt die Größe der Wellenfunktion, die von der Funktion zurückgegeben wird:

```
return ProdBottom + TopNode + remnantSum
```

Durch den *print*-Befehl wird das Ergebnis der Funktion ausgegeben:

```
print get_SPFs()
```

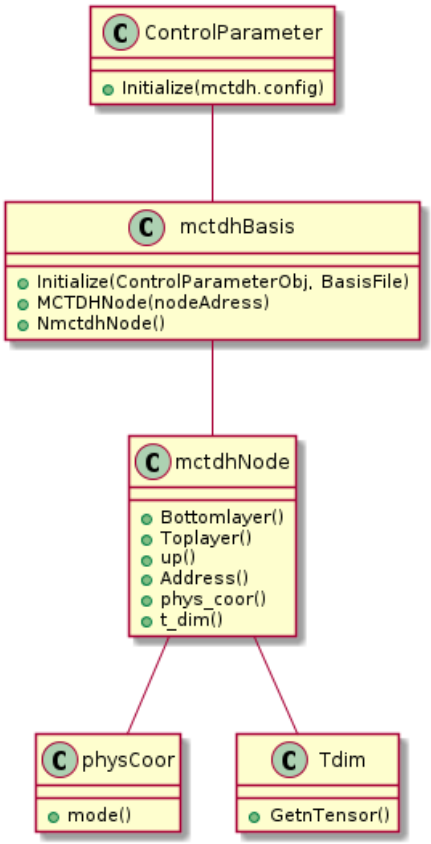

Abbildung 4.1.1: Alle Klassen, die in Cython erstellt wurden, sind mit einem „C" gekennzeichnet. Die jeweiligen Klassenmethoden sind mit einem grünen Punkt gekennzeichnet.

4.2 Graphische Benutzeroberfläche für MCTDH

Zur Demonstration der graphischen Benutzeroberfläche wird im folgenden das Vorgehen zum Starten einer MCTDH Simulation vorgeführt. Als Beispiel dient die Photodissociation von NOCl.

Bei Start der GUI erscheint das Hauptfenster (siehe Abbildung 4.2.1). In Liste **1** sind alle vorhandenen Projekte aufgeführt. Nach Auswahl eines Projekts erscheinen in Liste **2** alle dem Projekt zugeorednete Rechnungen. Durch die Schaltflächen „+" und „-" können Projekte sowie Rechnungen jeweils hinzugefügt bzw. entfernt werde. Im „File" **3** befinden sich zusätzliche Optionen zur Projektverwaltung (siehe Abbildung 4.2.2). Durch Klicken der Schaltfläche **5** wird ein neues Projekt erstellt. Externe Projekte werden durch die Schaltfläche **6** geladen. Betätigung der Schaltfläche **7** beendet die Benutzeroberfläche.

Zum Starten einer neuen Rechnung wird zuerst der Reiter "Project"(Abbildung 4.2.1 **4**) ausgewählt. Zwei neue Buttons erscheinen (Abbildung 4.2.3). Durch Klicken von **8** erscheint ein neues Fenster „MCTDH calculation" (siehe Abbildung 4.2.4). In diesem Fenster wird die neue MCTDH Rechnung spezifiziert.

In dem Eingabefenster „MCTDH calculation" kann im Feld **10** der Name der Rechnung angegeben werden. Sollte beim Speichern (Abbildung 4.2.4 **18**) des MCTDH-Baums und der Einstellungsparameter kein Name angegeben sein, wird der Benutzer aufgefordert einen Name für die Rechnung zu wählen. Bevor die Rechnung gespeichert wird, wird überprüft, ob der gewählte Name dem Namen einer andere Rechnung entspricht und, ob diese dann gegebenfalls überschrieben werden soll. In Liste **11** sind verschiedene molekulare Systeme aufgeführt. Über die Knöpfe **12** „on" und „off" wird gesteuert, ob eine Potentialfläche für die Rechnung verwendet werden soll. Ist der „off" Knopf aktiviert, werden keine Potentiale in der Liste **13** angezeigt. Bei der Standardeinstellung der GUI ist der „on"-Knopf aktiviert und es können Potentiale durch Klicken auf die Elemente der Liste **13** ausgewählt werden. Für die MCTDH-Simulation können vier verschiedene Rechnungen durchgeführt werden **14**: Eigenzustände, Real- und Imaginärzeitpropagation und Wärmeflusseigenzustände. Des Weiteren kann die Anfangszeit, die Endzeit, die Zeitschritte und die Anzahl an Iterationen für die Rechnung angegeben werden **15**. Alle Parameter werden automatisch geladen, indem eine System aus Liste **11** ausgewählt wird oder eine Eingabe-Datei durch die „Load" Schaltflälche **19** eingelesen wird. Des Weiteren wird der MCTDH-Baum schematisch **16** als Baumdiagramm angezeigt. Zusätzlich wird der MCTDH-Baum graphisch in **17** angezeigt. In **16** kann

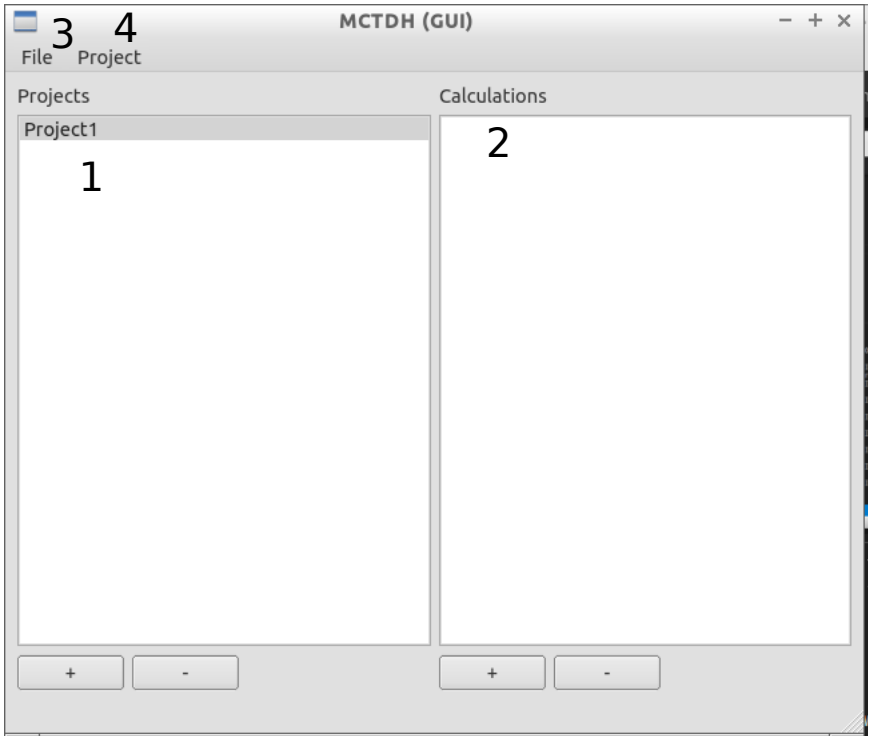

Abbildung 4.2.1: Hauptfenster der MCTDH-GUI

durch Doppelklick die Anzahl der SPFs, der primitiven Basis und der Moden verändert werden. Die Änderung wird instantan im Bild **17** angezeigt. Die Schaltfläche „Cancel" **21** beendet das Eingabefenster und die Schaltfläche „Start calculation" **20** beginnt die MCTDH Simulation. Alternativ kann die MCTDH Simulation auch aus dem Hauptfenster gestartet werden (siehe Abbildung 4.2.3. Durch Klicken der Schaltfläche „Run MCTDH on existing job" **9** wird ebenfalls die MCTDH Simulation gestartet. In Abbildung 4.2.5 wurde das System NOCl geladen, indem auf „NOCl" aus Liste **11** geklickt wurde.

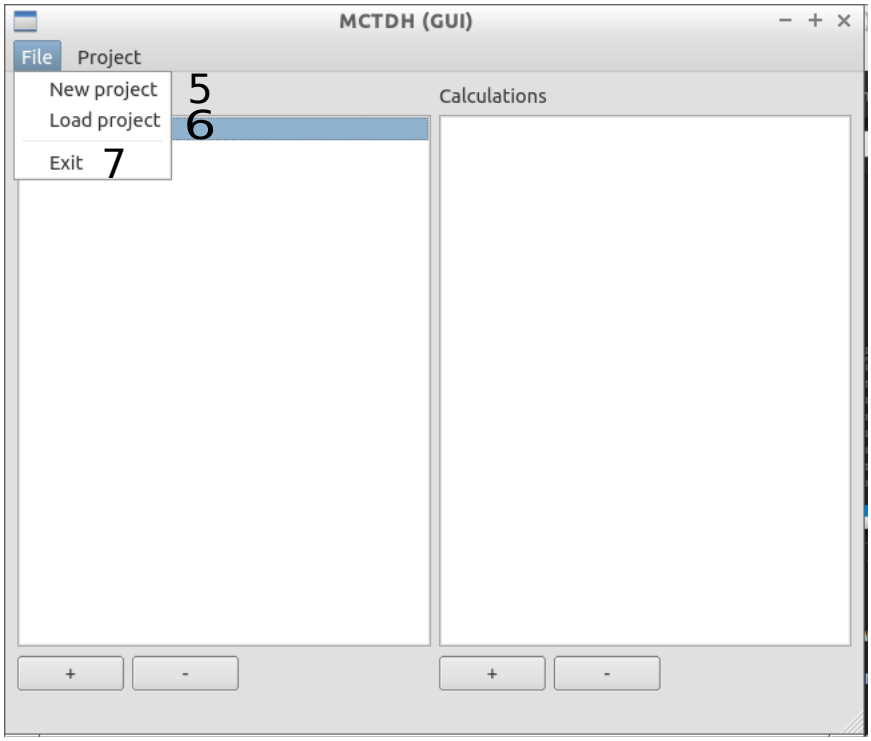

Abbildung 4.2.2: File-Menü des Hauptfenster der MCTDH-GUI

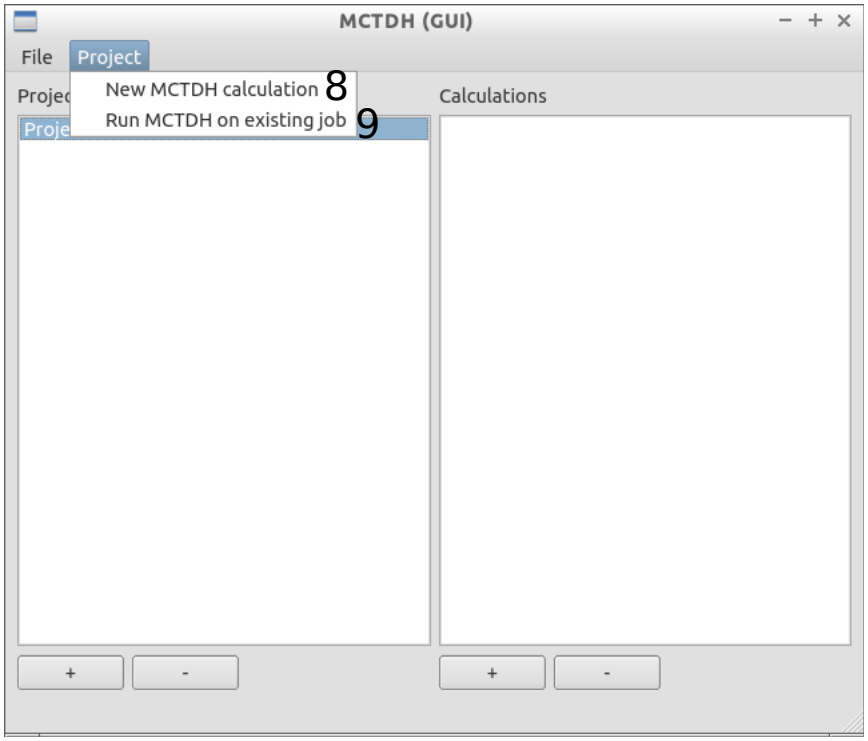

Abbildung 4.2.3: Eingabefenster der MCTDH-GUI

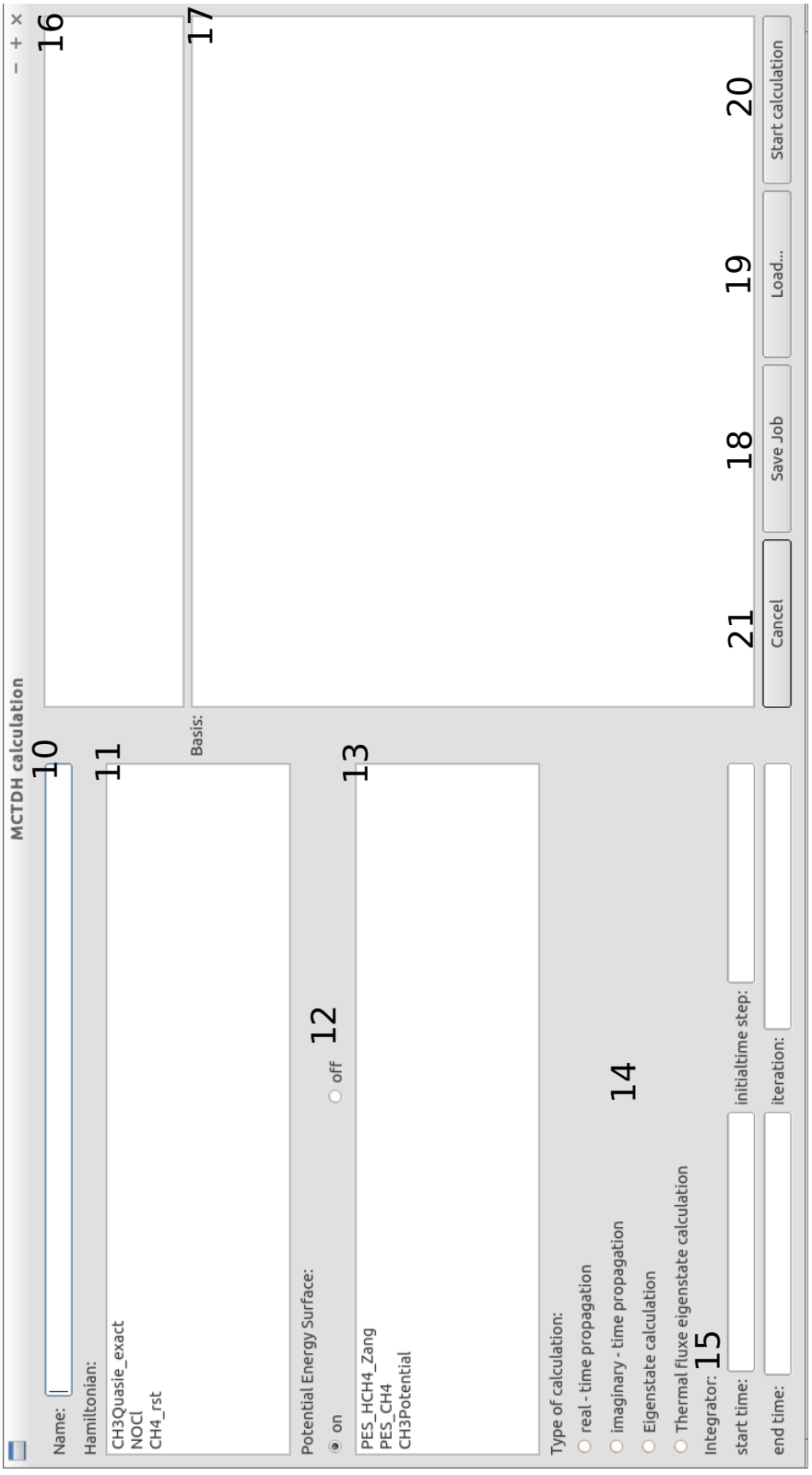

Abbildung 4.2.4: Eingabefenster der MCTDH-GUI

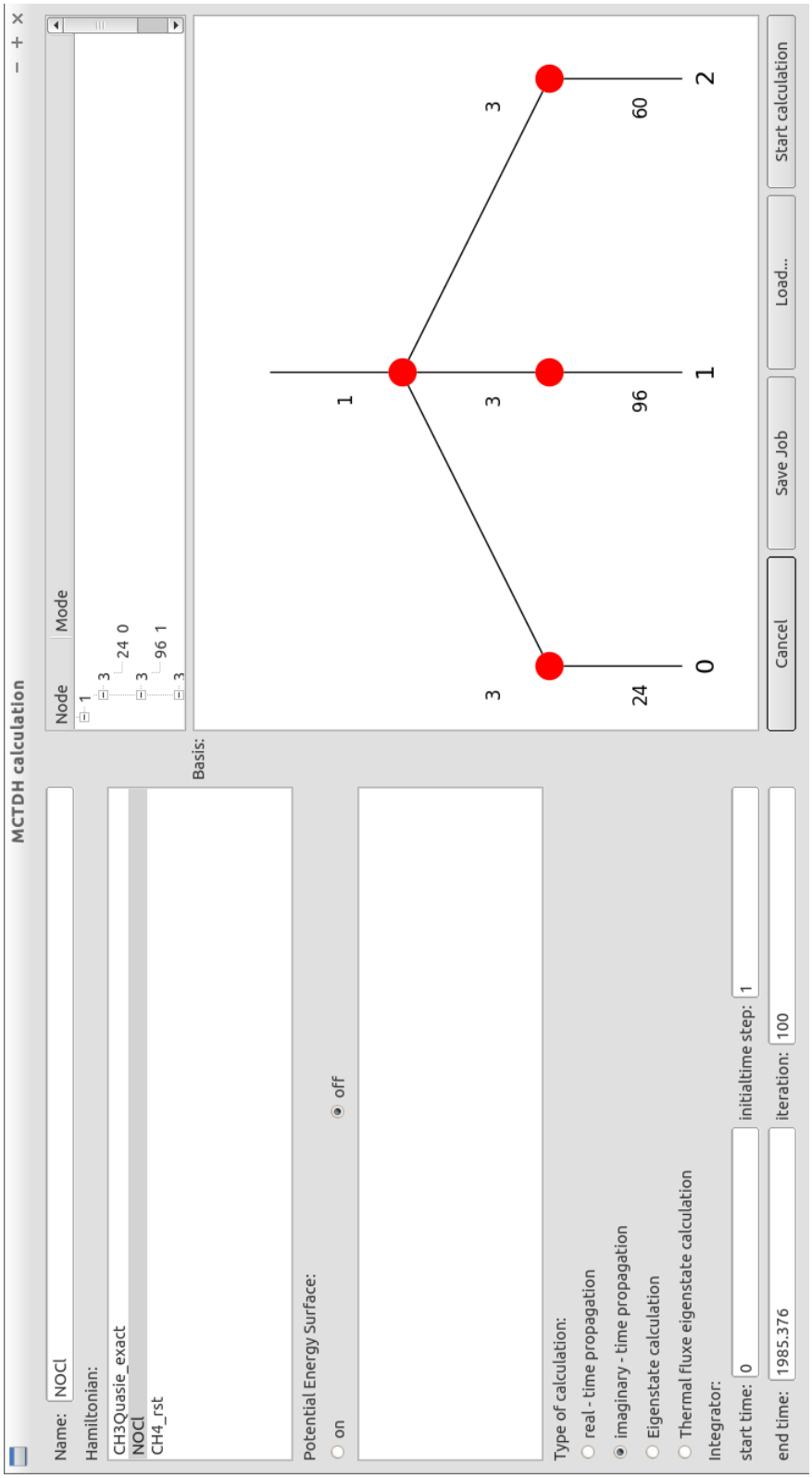

Abbildung 4.2.5: Eingabefenster der MCTDH-GUI am Beispiel der Photodissociation von NOCl.

5 Fazit und Ausblick

in dieser Arbeit wurde ein MCTDH-Python-Modul erstellt, das Klassen des MCTDH-Codes in Python nutzbar macht. In Zukunft könnten weitere Teile des MCTDH-Codes in Python zugänglich gemacht werden, sodass weitere Funktionen des MCTDH-Codes in Python verwendet werden ko"nnen.

Die MCTDH-GUI kann erweitert werden. Eine Auswahl an Basisdateien für verschiedene Hamiltonoperatoren existiert bereits. Der nächste Schritt wäre die Möglichkeit, dass der Benutzer die Hamiltonoperatoren durch das Anklicken vorgegebener Terme selber erstellt. Des Weiteren könnte ein Ausgabefenster erstellt werden, das die Zwischenergebnisse während der MCTDH-Rechnung graphisch darstellt. Die Drag&Drop-Funktion könnte für das Baumdiagramm in Abbildung 4.2.4 hinzugefügt werden, sodass die MCTDH-Bäum graphisch erstellt werden könnten. Schließlich wäre auch das Implementieren der MCTDH-GUI als App auf Smartphones für die Benutzer hilfreich, von dem sie die laufenden Rechnungen überprüfen könnten.

Literaturverzeichnis

[1] H.-D. Meyer, U. Manthe, and L. S. Cederbaum, Chem. Phys. Lett. **165**, 73 (1990).

[2] U. Manthe, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **97**, 3199 (1992).

[3] H. Wang and M. Thoss, J. Chem. Phys. **119**, 1289 (2003).

[4] U. Manthe, The Journal of Chemical Physics **128**, 164116 (2008).

[5] G. A. Worth, H. D. Meyer, H. Koeppel, L. S. Cederbaum, and I. Burghardt, Int. Rev. Phys. Chem. **27**, 569 (2008).

[6] T. Westermann, J. B. Kim, M. L. Weichman, C. Hock, T. I. Yacovitch, J. Palma, D. M. Neumark, and U. Manthe, Angew. Chem. Int. Ed. **53**, 1122 (2014).

[7] E. Y. Wilner, H. Wang, M. Thoss, and E. Rabani, Phys. Rev. B **89**, 205129 (2014).

[8] H. Wang, J. Phys. Chem. A **118**, 9253 (2014).

[9] K. Balzer, Z. Li, O. Vendrell, and M. Eckstein, Phys. Rev. B **91**, 045136 (2015).

[10] M. Schroeter and O. Kuehn, J. Phys. Chem. A **117**, 7580 (2013).

[11] M. Saab, M. Sala, B. Lasorne, F. Gatti, and S. Guerin, J. Chem. Phys. **141**, 134114 (2014).

[12] S. Lopez-Lopez, R. Martinazzo, and M. Nest, J. Chem. Phys. **134**, 094102 (2011).

[13] F. Bouakline, F. Lueder, R. Martinazzo, and P. Saalfrank, J. Phys. Chem. A **116**, 11118 (2012).

[14] L. Uranga-Pina, C. Meier, and J. Rubayo-Soneira, Chem. Phys. Lett. **543**, 12 (2012).

[15] M. Moix Teixidor and F. Huarte-Larranaga, Chem. Phys. **399**, 264 (2012).

[16] J. Wahl, R. Binder, and I. Burghardt, Comp. Theo. Chem. **1040**, 167 (2014).

[17] J. M. Schurer, P. Schmelcher, and A. Negretti, Phys. Rev. A **90**, 033601 (2014).

[18] V. S. Reddy, C. Camacho, J. Xia, R. Jasti, and S. Irle, J. Chem. Theo. Comp. **10**, 4025 (2014).

[19] W. Eisfeld, O. Vieuxmaire, and A. Viel, J. Chem. Phys. **140**, 224109 (2014).

[20] A. Valdes and R. Prosmiti, J. Phys. Chem. A **117**, 9518 (2013).

[21] T. Mondal, S. R. Reddy, and S. Mahapatra, J. Chem. Phys. **137**, 054311 (2012).

[22] D. Skouteris and A. Lagana, Chem. Phys. Lett. **575**, 18 (2013).

[23] B. Zhao, D.-H. Zhang, S.-Y. Lee, and Z. Sun, J. Chem. Phys. **140**, 164108 (2014).

[24] M. D. Coutinho-Neto, A. Viel, and U. Manthe, J. Chem. Phys. **121**, 9207 (2004).

[25] T. Hammer, M. D. Coutinho-Neto, A. Viel, and U. Manthe, J. Chem. Phys. **131**, 224109 (2009).

[26] T. Hammer and U. Manthe, J. Chem. Phys. **134**, 224305 (2011).

[27] M. Schroeder, F. Gatti, and H.-D. Meyer, J. Chem. Phys. **134**, 234307 (2011).

[28] T. Hammer and U. Manthe, J. Chem. Phys. **136**, 054105 (2012).

[29] M. Schroeder and H.-D. Meyer, J. Chem. Phys. **141**, 034116 (2014).

[30] O. Vendrell, F. Gatti, D. Lauvergnat, and H.-D. Meyer, Angew. Chemie Int. Ed. **46**, 6918 (2007).

[31] O. Vendrell, F. Gatti, D. Lauvergnat, and H.-D. Meyer, J. Chem. Phys. **127**, 184302 (2007).

[32] O. Vendrell, F. Gatti, and H.-D. Meyer, J. Chem. Phys. **127**, 184303 (2007).

[33] O. Vendrell, M. Brill, F. Gatti, and H.-D. Meyer, J. Chem. Phys. **130**, 234305 (2009).

[34] O. Vendrell, F. Gatti, and H.-D. Meyer, J. Chem. Phys. **131**, 034308 (2009).

[35] F. Huarte-Larrañaga and U. Manthe, J. Chem. Phys. **113**, 5115 (2000).

[36] F. Huarte-Larrañaga and U. Manthe, J. Phys. Chem. A **105**, 2522 (2001).

[37] T. Wu, H.-J. Werner, and U. Manthe, Science **306**, 2227 (2004).

[38] G. Schiffel and U. Manthe, J. Chem. Phys. **132**, 084103 (2010).

[39] R. van Harrevelt, G. Nyman, and U. Manthe, J. Chem. Phys. **126**, 084303 (2007).

[40] G. Nyman, R. van Harrevelt, and U. Manthe, J. Phys. Chem. A **111**, 10331 (2007).

[41] G. Schiffel and U. Manthe, J. Chem. Phys. **132**, 191101 (2010).

[42] G. Schiffel and U. Manthe, J. Chem. Phys. **133**, 174124 (2010).

[43] R. Welsch and U. Manthe, J. Chem. Phys. **141**, 051102 (2014).

[44] R. Welsch and U. Manthe, J. Chem. Phys. **141**, 174313 (2014).

[45] R. Welsch and U. Manthe, J. Chem. Phys. **142**, 064309 (2015).

[46] R. Welsch and U. Manthe, J. Phys. Chem. Lett. **6**, 338 (2015).

[47] G. A. Worth, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **105**, 4412 (1996).

[48] G. A. Worth, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **109**, 3518 (1998).

[49] A. Raab, G. A. Worth, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **110**, 936 (1999).

[50] H. Wang, D. E. Skinner, and M. Thoss, J. Chem. Phys. **125**, 174502 (2006).

[51] I. Kondov, M. Cizek, C. Benesch, M. Thoss, and H. Wang, J. Phys. Chem. C **111**, 11970 (2007).

[52] I. R. Craig, M. Thoss, and H. Wang, J. Chem. Phys. **135**, 064504 (2011).

[53] H. Wang, I. Pshenichnyuk, R. Härtle, and M. Thoss, J. Chem. Phys. **135**, 244506 (2011).

[54] T. Westermann, R. Brodbeck, A. B. Rozhenko, W. W. Schoeller, and U. Manthe, J. Chem. Phys. **135**, 184102 (2011).

[55] M. H. Beck, A. Jäckle, G. A. Worth, and H.-D. Meyer, Physics Reports **324**, 1 (2000).

[56] H.-D. Meyer and G. A. Worth, Theor. Chem. Acc. **109**, 251 (2003).

[57] F. Huarte-Larrañaga and U. Manthe, Z. Phys. Chem. **221**, 171 (2007).

[58] H.-D. Meyer, F. Gatti, and G. A. Worth, *Multidimensional Quantum Dynamics: MCTDH Theory and Applications* (Weinheim: Wiley-VCH, 2009).

[59] U. Manthe, Mol. Phys. **109**, 1415 (2011).

[60] H.-D. Meyer, Wiley Interdisciplinary Reviews: Computational Molecular Science **2**, 351 (2012).

[61] *An open source machine learning framework for everyone* (2018), URL https://www.tensorflow.org/.

[62] *Scientific computing tools for python* (2018), URL https://scipy.org/about.html.

[63] F. Huarte-Larrañaga and U. Manthe, J. Chem. Phys. **116**, 2863 (2002).

[64] T. Wu, H.-J. Werner, and U. Manthe, J. Chem. Phys. **124**, 164307 (2006).

[65] F. Huarte-Larrañaga and U. Manthe, J. Chem. Phys. **117**, 4635 (2002).

[66] J. M. Bowman, D. Wang, X. Huang, F. Huarte-Larrañaga, and U. Manthe, J. Chem. Phys. **114**, 9683 (2001).

[67] C. Cattarius, G. A. Worth, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **115**, 2088 (2001).

[68] H. Wang, J. Chem. Phys. **113**, 9948 (2000).

[69] H. Wang, M. Thoss, and W. Miller, J. Chem. Phys. **115**, 2979 (2001).

[70] M. Nest and H.-D. Meyer, J. Chem. Phys. **119**, 24 (2003).

[71] U. Manthe, J. Chem. Phys. **105**, 6989 (1996).

[72] R. van Harrevelt und U. Manthe, J. Chem. Phys. **121**, 5623 (2004).

[73] R. van Harrevelt and U. Manthe, J. Chem. Phys. **123**, 064106 (2005).

[74] M. Summerfield, *Rapid GUI Programming with Python and Qt* (Prentice Hall, 2008).

[75] *Model/view programming* (2018), URL http://doc.qt.io/qt-5/model-view-programming.html.

[76] G. Krasner and S. Pope, Tech. Rep., ParcPlace Systems, Mountain View, Calif. (2002).