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

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 berechnet. 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, da ein tiefgreifendes Verständnis der Programmstruktur erforderlich ist. Im Rahmen dieser Masterarbeit wurden zwei wesentliche Verbesserungen unternommen: Es wurde eine Benutzeroberfläche (von englisch graphical user interface,

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 ein tiefgreifendes Verständnis der Programmstruktur vorrauszusetzen. 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] usw.).

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

2 Theorie

Die Effizienz des MCTDH-Verfahrens resultiert aus der doppellagigen Struktur der verwendeten Wellenfunktion. In anderen Wellenpaktdynamikverfahren, wie der Standardmethode[60], wird die Wellenfunktion direkt in einer zeitunabhängigen Basis oder einem zeitunabhängigen Gitter dargestellt. Anstatt 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 (von englisch single-particle function, SPF) genannt. Die SPFs werden in der primitiven zeitunabhängigen Basis oder dem 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, welche die zeitabhängigen SPFs in der primitiven zeitunabhängigen Basis darstellen, bilden die untere Lage. Die Bewegungsgleichungen, durch die gleichzeitig die optimalen Entwicklungskoeffizienten für beide Lagen bestimmt werden, ergeben sich aus dem Dirac-Frenkel Variationsprinzip.

Dennoch ist auch das MCTDH-Verfahren 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 noch immer 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 mindestens 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 Moden-Kombinationsverfahren behandelt werden. Zwar konnte durch dieses Verfahren die Grenze zu höherer Dimensionalität verschoben werden, dennoch bleibt die grundlegende Einschränkung: Der numerische Aufwand skaliert mindestens mit 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, welche 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 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 Verfahren 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 modenkombinierten 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 Ansätze der 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. in einem zeitunabhängigen Gitter dargestellt. Die mehrdimensionale Wellenfunktion wird durch das direkte Produkt von eindimensionalen Basisfunktionen $\mathcal{X}_j^\kappa(x_\kappa)$ 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 $\phi^{1;\kappa}_j(x_\kappa, t)$ entwickelt.

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

Anschließend werden diese SPFs innerhalb der zeitunabhängigen primitive Basis dargestellt:

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

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 $z = 2$, dass 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-ten Koordinate.

Zur Visualisierung unterschiedlicher MCTDH-Wellenfunktionen, für die unterschiedlich viele Lagen verwendet wurden, dienen Baumdiagramme, wie sie in Abbildung 2.1.1 gezeigt werden. 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 Koeffizient $A^{2;1}_{m;j}$, der durch die SPFs $\phi^{1;1}_{j_1}(x_1, t)$ 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 zeitunabhängigen 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 Hochzahlen 1 und 2 in dieser Notation zeigen, 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:

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

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

In Abbildung 2.1.1c ist das entsprechende Diagramm der Wellenfunktion eines siebendimensionalen Systems dargestellt, dessen Koordinaten in logischen Koordinaten zusam-

mengefasst 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 $q_3^1 = \{x_5, x_6, x_7\}$.

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, 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

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

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

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

In Gleichung 2.7 ist $\phi_m^{2;\kappa,\lambda}$ mit der dazugehörigen Koordinate $q_\lambda^{2;\kappa}$ 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)

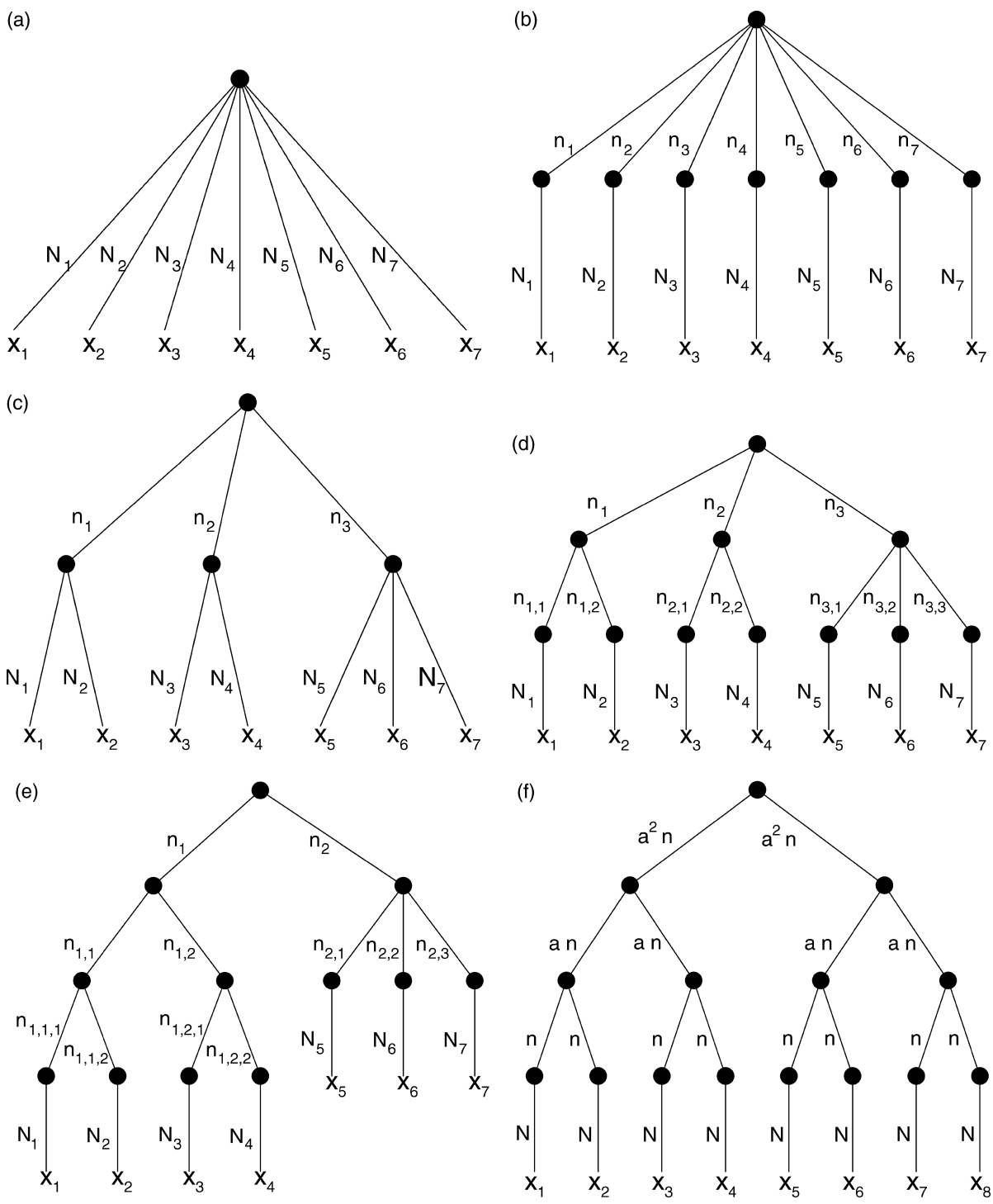

Abbildung 2.1.1: Unterschiedliche Darstellungen von Wellenfunktionen eines siebendimensionalen Systems. Dargestellt sind: (a) Eine Darstellung eines Standardwellenpakets, (b) eine MCTDH-Wellenfunktion, (c) eine modenkombinierte MCTDH-Wellenfunktion, [(d)-(f)] eine Multilayer-MCTDH-Wellenfunktion. Die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

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 Details

Die Python-Schnittstelle für MCTDH wurde in der Programmiersprache Cython erstellt. Cython stellt eine Hybridprogrammiersprache dar, die Python und C\C++ kombiniert. Im Gegensatz zu Python wird Cython kompiliert, wobei die Cythonsyntax der Pythonsyntax ähnlich ist. In Cython kann C++ Code verwendet werden, der beim Erstellen der Cython-Programme ebenfalls kompiliert wird. Auf diese Weise wurden mehrere *mctdh*-Klassen auf Basis von C++ Klassen erstellt.

Die GUI für die MCTDH-Rechnungen wurde in Python und Qt implementiert. Der Zugriff auf die Qt-Bibliothek erfolgt über die Python-Bibliothek PyQt4. Das Design der einzelnen GUI-Fenster erfolgte in Qt-Designer. Die Informationen über die jeweiligen Fenster werden in *ui*-Dateien gespeichert. Mit PyQt können diese Dateien eingelesen werden und die entsprechenden Klassen erstellt und beliebig erweitert werden. Die in Qt-Designer erstellten Fenster werden in PyQt miteinander verknüpft.

Neben PyQt wurden die Python-Module *networkx* und *matplotlib* verwendet. Mithilfe von Klassen aus *networkx* und *matplotlib* konnten Baumdiagramme erstellt und gespeichert werden, die in der GUI zur Visualisierung des MCTDH-Baums verwendet wurden.

4 Ergebnisse

4.1 Python-Interface für MCTDH

Es wurde eine Programmierschnittstelle (von englisch *application programming interface*, API) für das MCTDH-Programmpaket erstellt, welche es erlaubt Klassen und Funktionen des in C++ verfassten MCTDH-Codes in Python-Skripten zu verwenden. Die Schnittstelle umfasst Klassen und Methoden, die zur Verwaltung der MCTDH-Basis dienen. Die Erweiterung auf andere Klassen kann nach einem einfachen Schema erfolgen. Abbildung 4.1.1 zeigt ein Klassendiagramm aller Klassen, welche über die API aufgerufen werden können. Jede Klasse der API wird in Abbildung 4.1.1 durch ein Rechteck repräsentiert. Im oberen Teil des Rechtecks ist der Klassenname angegeben und im unteren Teil sind die Methoden der Klasse aufgelistet. Die baumartige Anordnung der Rechtecke repräsentiert die Abhängigkeit der Klassen zueinander. Eine Klasse hängt von allen Klassen ab, die sich im Diagramm unterhalb der betrachteten Klasse befinden und zu denen eine Verbindung besteht. In der Klasse *Controlparameters* werden die Genauigkeitsparameter der MCTDH-Rechnung verwaltet. Die Klasse *physCoor* verwaltet Informationen einer Koordinate und enthält das primitive Gitter oder die primitive Basis. Sie ist ein Memberobjekt der Klasse *mctdhNode*, welche einen Knoten im MCTDH-Baum repräsentiert. In der Klasse *mctdhNode* wird die lokale Konnektivität des Knoten, sowie dessen Satz von Entwicklungskoeffizienten verwaltet. Die Dimensionen der Entwicklungskoeffizienten sind in Objekten der Klasse *Tdim* gespeichert. Die Klasse *mctdhBasis* enthält alle Objekte, die dem Management der MCTDH-Basis dienen.

Zur Demostration der API wird im folgenden ein Python-Skript vorgestellt, mithilfe dessen Dimensionsinformationen einer Multilayer-MCTDH-Wellenfunktion berechnet werden können. Zuerst wird das Skript vorgestellt und anschließend wird es abschnittsweise erklärt.

```
import mctdh

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

maxNodes = basis.NmctdhNodes()

nodes_spf = {}
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}
NCoefBottomNode = {}
for key in primitivB:
    NCoefBottomNode[key] = primitivB[key] * nodes_spf[key]
NCoefBottom = sum([l_[1] for l_ in NCoefBottomNode.items()])

print NCoefBottom

NCoefTopNode = 0
for i in range(maxNodes):
    if basis.MCTDHnode(i).Toplayer() == True:
            children = basis.MCTDHnode(i).NChildren()
            for j in range(children):
                NCoefTopNode *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
            NCoefTopNode *= basis.MCTDHnode(i).t_dim().GetnTensor()

print NCoefTopNode
```

```
remnantNodeList = []
remnant = 0
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
NCoefRemnant = sum(remnantNodeList)

print NCoefRemnant
```

Die in Abbildung 4.1.1 dargestellten Klassen werden mit folgenden Befehl in Python importiert:

```
import mctdh
```

Objekte der Klassen *ControlParameters* und *mctdhBasis* werden instanziert durch:

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

Die erstellten Objekte werden mittels

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

initialisiert, wobei *mctdh.config* und *basis.txt* Dateinamen von Dateien sind, welche die Genauigkeitsparameter, bzw. die Basis-Definition enthalten.

Die Anzahl der Knoten im MCTDH-Baum wird mit dem folgenden Befehl bestimmt:

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

Danach wird die Anzahl der Tensoren für jeden Knoten bestimmt. Die entsprechenden Dimensionen sind Member der Klasse *Tdim*, welche in den repräsentativen Knotenobjekten gespeichert sind. Sie werden ermittelt und in dem Dictionary *nodes_spf* gespeichert:

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

Eine Liste der Größen aller primitiven Basissätze wird durch die folgenden Befehle erzeugt:

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

Daraufhin wird die Anzahl aller Entwicklungskoeffizienten der untersten Lage berechnet durch

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

Die Anzahl der Entwicklungskoeffizienten der obersten Lage wird durch den 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):
                NCoefTopNode *= basis.MCTDHnode(i).down(j).t_dim().GetnTensor()
            NCoefTopNode *= basis.MCTDHnode(i).t_dim().GetnTensor()
```

Im letzten Code-Block wird die Summe der Anzahl der Entwicklungskoeffizienten aus den übrigen Lagen 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
    NCoefRemnant = sum(remnantNodeList)
```

Schließlich wird die Anzahl aller Entwicklungskoeffizienten der unteren Knoten, des oberen Knoten und der restlichen Knoten berechnet und ausgegeben:

```
print NCoefBottom
print NCoefTopNode
print NCoefRemnant
```

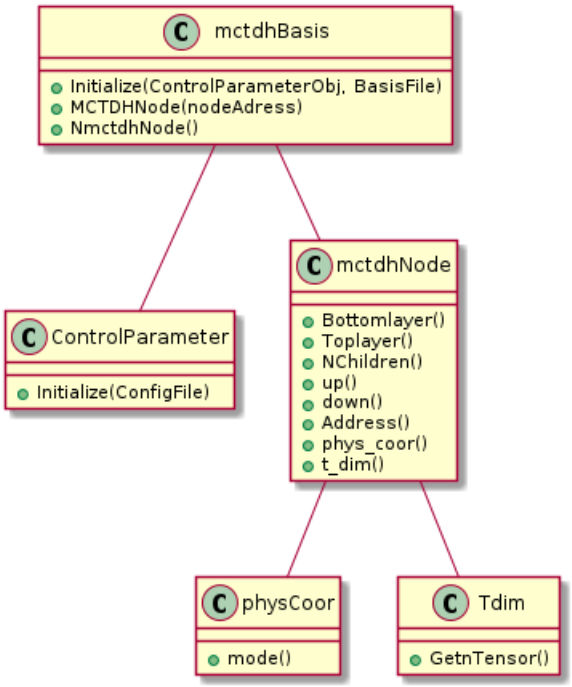

Abbildung 4.1.1: Klassendiagramm der Programmierschnittstelle (API). Die Rechtecke repräsentieren jeweils eine Klasse. Im oberen Bereich des Rechtecks befindet sich der Name der Klasse und im unteren Bereich eine Liste aller Memberfunktionen. Für eine detaillierte Beschreibung siehe Text.

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 Realzeitpropagation eines Wellenpakets auf dem ersten elektronisch angeregten Zustand (S_1) von NOCl.

Beim 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 zugeoredneten Rechnungen. Durch die Schaltflächen „+" und „-" können Projekte sowie Rechnungen jeweils hinzugefügt bzw. entfernt werde. Im Reiter „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 mit dem Namen „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. Beim Speichern der Rechnung wird überprüft, ob eine Rechnung mit dem ausgewählten Namen existiert, um ein unabsichtliches Überschreiben zu verhindern. In Liste **11** wird der entsprechende Summe-von-Produkten Operator für das gegebene System ausgewählt. Über die Knöpfe „on" und „off" (**12**) wird gesteuert, ob eine Potentialflächenauswertung mittels CDVR erfolgt. Ist der „off" Knopf aktiviert, werden keine Potentiale in der Liste **13** angezeigt. Bei der Standardeinstellung der GUI ist der Knopf „on" aktiviert und es können Potentiale durch Klicken auf die Elemente der Liste **13** ausgewählt werden. Im vorliegenden Beispiel wird die Option „off" ausgewählt. Die Erweiterung der Summe-von-Produkten Operatoren und Potentialflächen kann schematisch erfolgen.

Mithilfe der MCTDH GUI können vier verschiedene Operationen durchgeführt werden (**14**): „real-time propagation", „imaginary-time propagation", „Eigenstate calculation" und „Thermal flux eigenstate calculation". Hier wird die Option „real-time propagation"

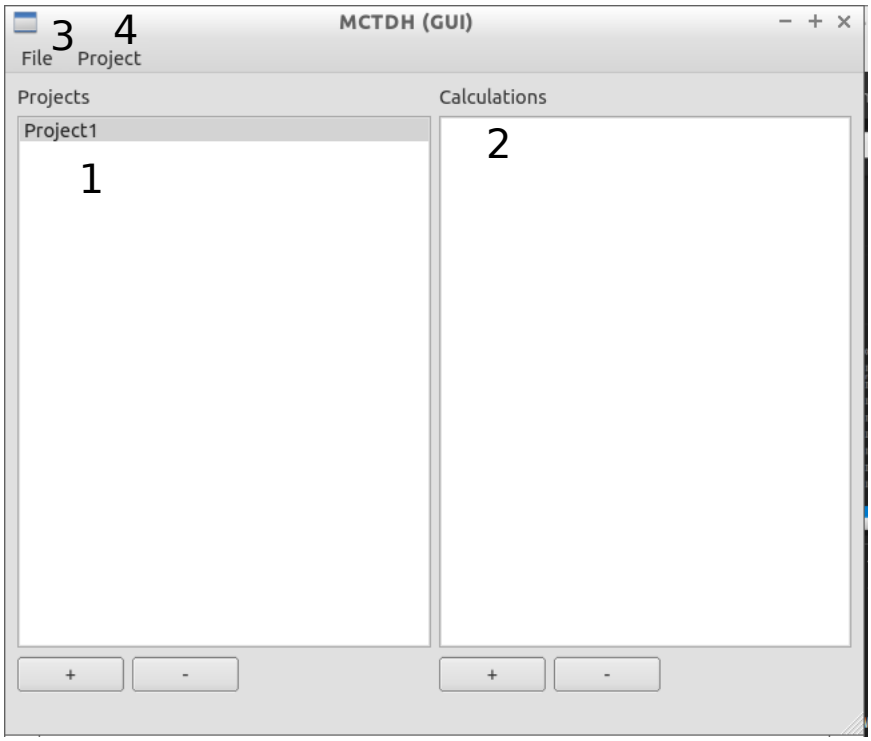

Abbildung 4.2.1: Hauptfenster der MCTDH-GUI. Es zeigt aktuell vorhandene Projekte (**1**), sowie zugehörige Rechnungen (**2**).

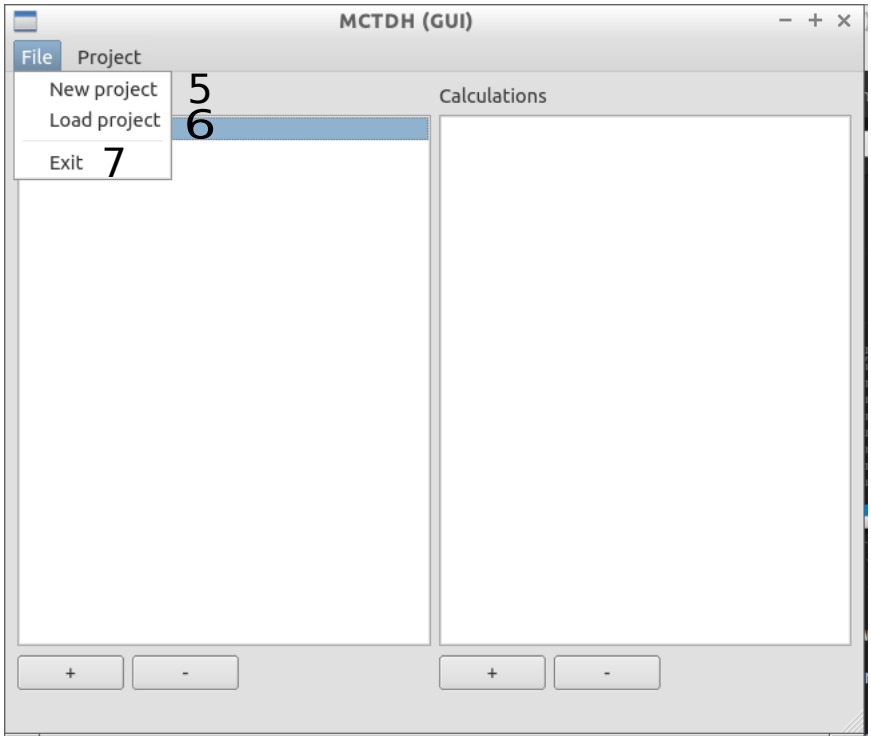

Abbildung 4.2.2: Hauptfenster der MCTDH-GUI. Nach Betätigung des Reiters „File" stehen Optionen zur Projektverwaltung, sowie zum Beenden der GUI zur Verfügung.

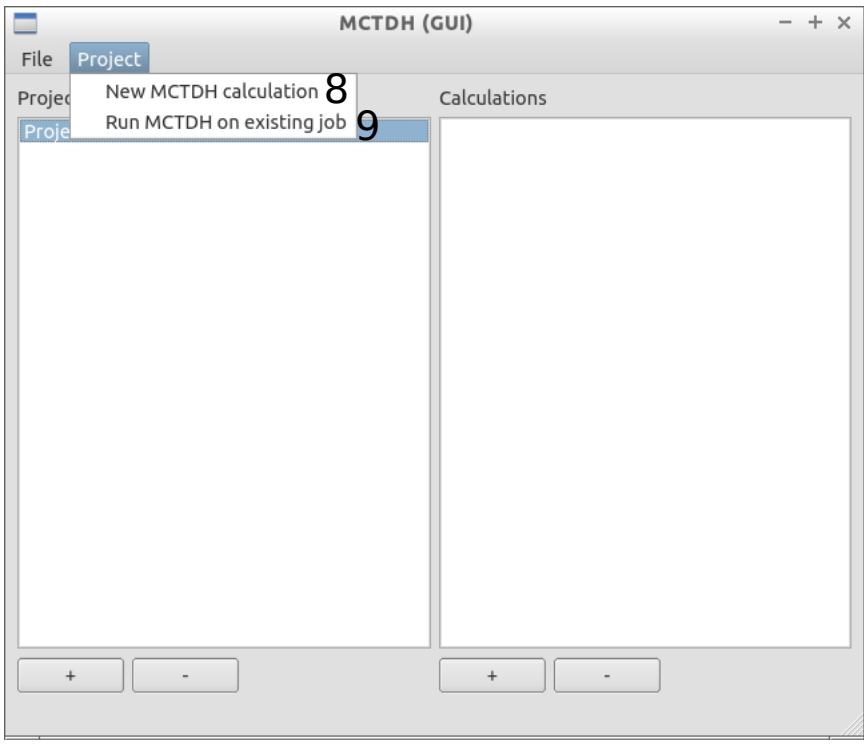

Abbildung 4.2.3: Hauptfenster der MCTDH-GUI nach Betätigung des Reiters „Project". Es stehen Optionen zum Erstellen und Starten einer MCTDH-Rechnung zur Verfügung.

ausgewählt. Des Weiteren werden unter **15** Steuerungsparameter für den Integrator ausgewählt. Dazu gehören die Anfangszeit, die Endzeit, der initiale Zeitschritt und (für die Operationen „Eigenstate calculation" und „Thermal flux eigenstate calculation") die Anzahl an Iterationen. Alle Parameter werden automatisch geladen, indem ein System aus Liste **11** ausgewählt wird oder eine Eingabe-Datei durch die „Load" Schaltflälche **19** eingelesen wird. Der MCTDH-Baum, welcher die aktuell gewählte Basis repräsentiert, wird unter **16** als Listendiagramm angezeigt. Hier kann durch Doppelklick die Anzahl der SPFs, der primitiven Basis und der Moden verändert werden. Zusätzlich wird der MCTDH-Baum in Form eines Diagramms in **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, indem Schaltfläche „Run MCTDH on existing job" (**9**) betätigt wird (siehe Abbildung 4.2.3).

In Abbildung 4.2.5 wurde das System NOCl mit sinnvollen Eingabeparametern geladen, indem auf „NOCl" aus Liste **11** geklickt wurde.

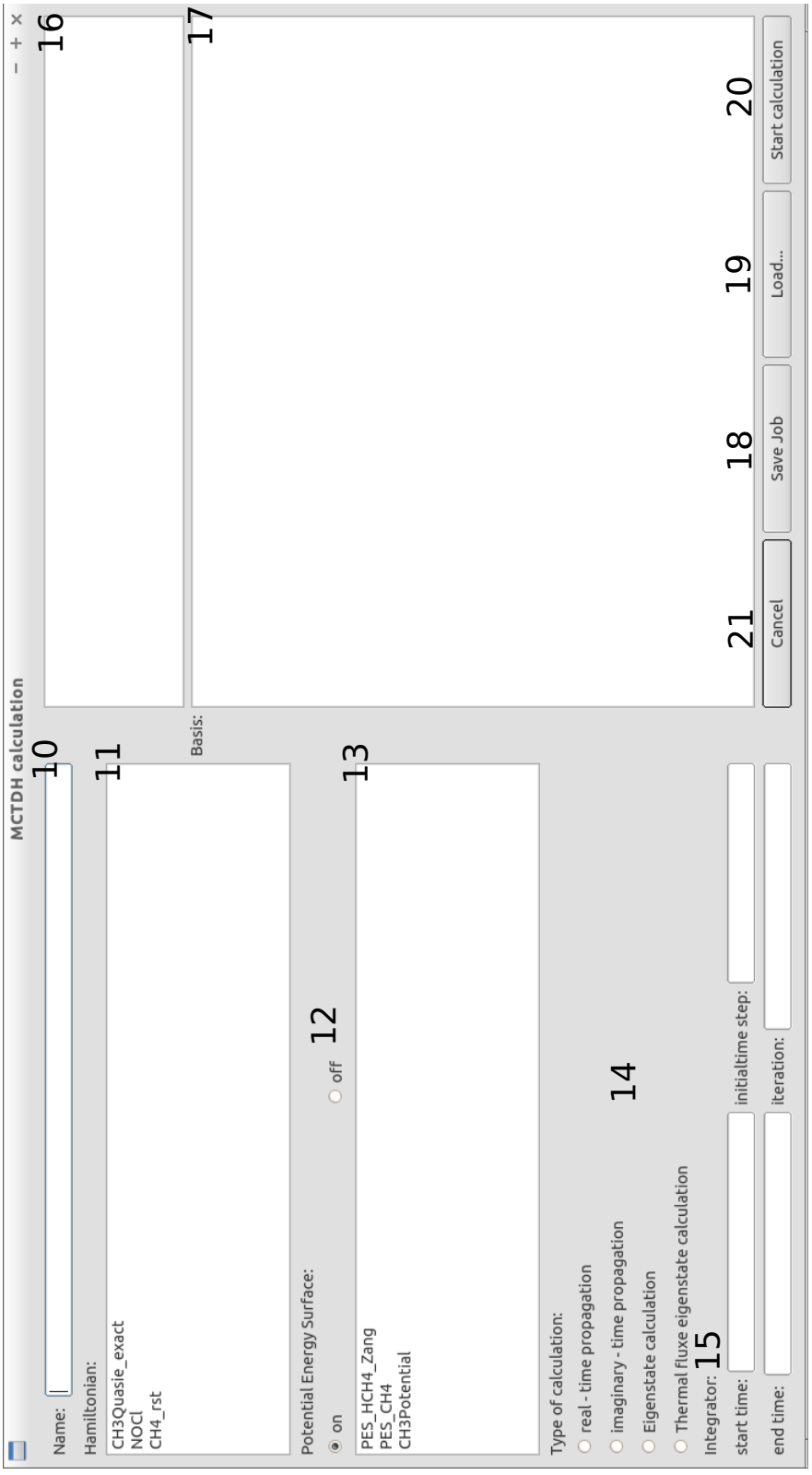

Abbildung 4.2.4: Spezifikation einer neuen MCTDH-Rechnung durch die MCTDH-GUI.

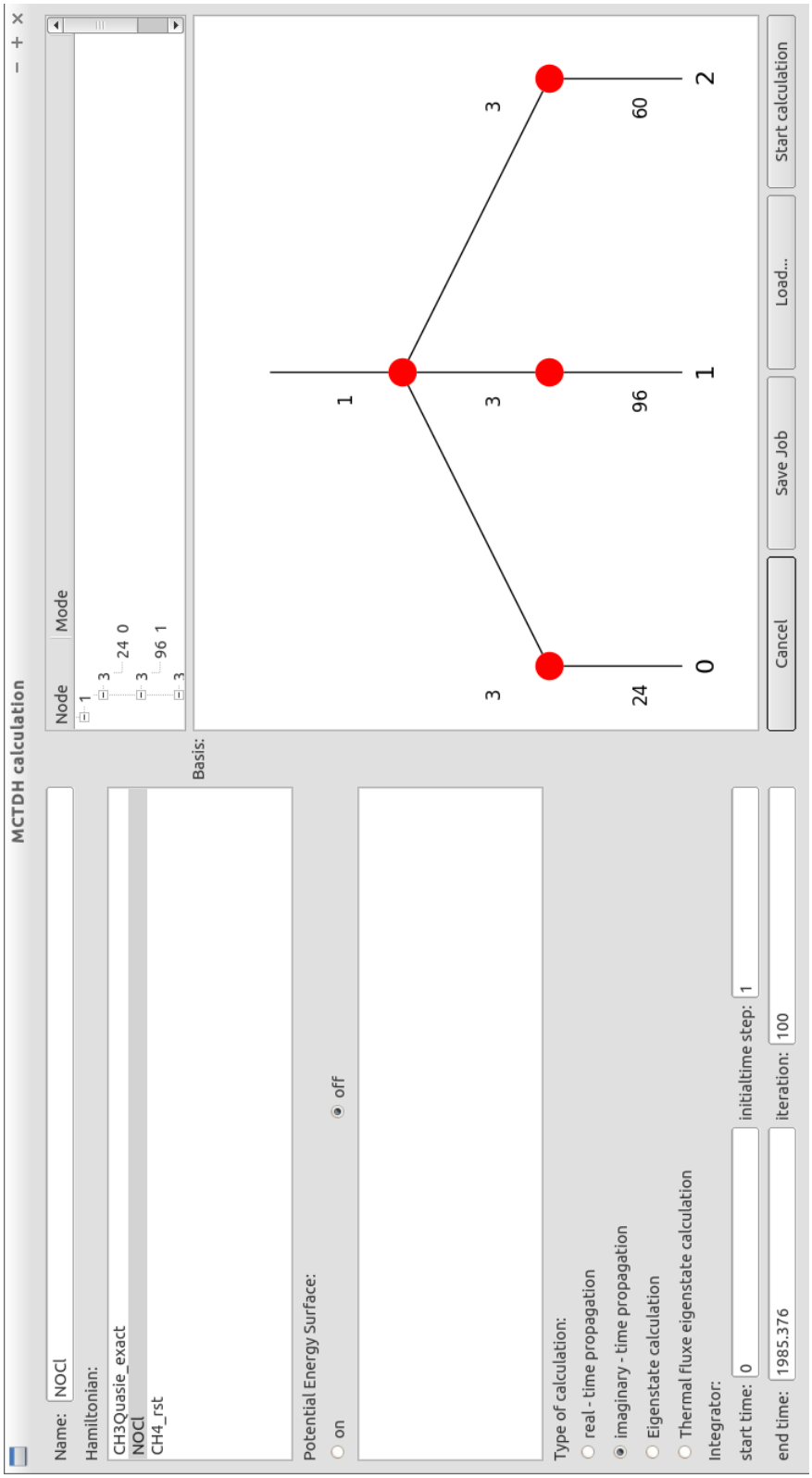

Abbildung 4.2.5: Eingabefenster der MCTDH-GUI am Beispiel der Photodissoziation von NOCl.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Pythonschnittstelle für den in C++ verfassten MCTDH-Code erstellt, welche es erlaubt Klassen in Python-Skripten zu nutzen, die der Verwaltung der MCTDH-Basis dienen. In Zukunft kann die Schnittstelle um weitere Klassen des MCTDH-Codes ergänzt werden, sodass sie vielseitig eingesetzt werden kann.

Eine graphische Benutzeroberfläche wurde erstellt, welche es Nutzern ohne Programmierkenntnisse erlaubt MCTDH-Rechnungen zu starten. Im Rahmen dieser Arbeit wurde in der GUI eine Auswahl von Summe-von-Produkten Operatoren, sowie von *ab initio* Potentialflächen bereitgestellt. Diese Auswahl kann in Zukunft erweitert werden. Zudem wäre eine Funktion zum Erstellen von benutzerdefinierten Operatoren mittels Drag&Drop hilfreich. Die Struktur der MCTDH-Basis wird aktuell über externe Dateien eingelesen; auch hier wäre es sinnvoll eine Funktion zum Erstellen und Modifizieren der Multilayer-MCTDH Basisstruktur mittels Drag&Drop bereitzustellen.

Die Funktionen der GUI konzentriert sich bisher auf das Erstellen und Verwalten von projektbezogenen Rechnungen. Es wäre nützlich die Funktionen auf das Verfolgen und Visualisieren laufender oder beendeter Rechnungen auszuweiten. Dazu ist es sinnvoll für die Ausgabe des C++ MCTDH-Programms ein standardisiertes Protokollformat zu entwickeln, welches von externen Programmen eingelesen und interpretiert werden kann. Somit wäre beispielsweise ein Einbinden von Smartphones und Tabletts denkbar, um laufende Rechnungen unkompliziert und leicht verfolgen zu können.

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).