

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 ml-MCTDH-Wellenfunktion	5
3	Fazit und Ausblick	11
4	Ergebnisse	12
4.1	Python-Interface für MCTDH	12
4.2	Graphische Benutzeroberfläche für MCTDH	17
	Literaturverzeichnis	21

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 wurde. 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-Rechnungen mit Modelhamiltonoperatoren untersucht. So wurde in wegweisenden Rechnungen die nichtadibatischen Dynamiken von Pyrazin erforscht, in denen ein 24-moden schwingungsgekoppelter Hamiltonoperator^[47–49] verwendet wurde. Multilayer-MCTDH Simulationen von typischen physikalischen Modellen^[3,8,50–53] zu kondensierter Materie schließen üblich tausende Freiheitsgrade ein. Für die Untersuchung eines Photodissoziationsmodel wurden in einem Wirt-Gast-Komplex 189- dimensionale ml-MCTDH Rechnungen durchgeführt^[54]. Für weitere Literatur zum MCTDH-Verfahren und seine Anwendungen diskutieren, siehe Refs.^[55–60].

Das MCTDH-Programmpaket, welches zur Berechnung und Simulation der oben genannten Systeme wird, bietet die Möglichkeit zur hocheffizienten Propagation von Wellenfunktionen und der Berechnung von Dichtematrizen und Eigenzuständen. Bisher ist die Bedienung des vorliegenden MCTDH-Programmpakets selbst bei Standardaufgaben Spezialisten vorenthalten und ein tiefgreifendes Verständnis des Codes ist erforderlich. Im Rahmen dieser Masterarbeit wurden zwei wesentliche Verbesserungen unternommen: Zum einen wurde eine Benutzeroberfläche (GUI) erstellt, die Wissenschaftlern ohne Pro-

grammierkenntnissen 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.

Python hat sich als einsteigerfreundliche Programmiersprache erwiesen. Für viele häufig genutzte und komplexe numerische Programm-Codes existiert bereits ein in Python verfasstes Programminterface, welches die Nutzung des Codes ermöglicht, ohne dass die Programmstruktur des zugrundeliegenden Programms verstanden werden muss (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 ml-MCTDH herausgestellt. In Kapitel 3 wird die Python-Schnittstelle und die graphischen Benutzeroberfläche beschrieben. Schließlich wird in Kapitel 4 eine Fazit gezogen und ein Ausblick gegeben.

2 Theorie

Die Effizienz des MCTDH-Verfahrens resultiert aus der Doppellayerstruktur der verwendeten Wellenfunktion. In anderen Wellenpaketendynamikverfahren, wie der Standardmethode^[60] wird die Wellenfunktion direkt in einer zeitunabhängigen Basis oder einem zeitunabhängigen Gitter entwickelt. Anstelle die Wellenfunktion in einer zeitunabhängigen Basis zu entwickeln, wird im MCTDH-Verfahren das korrelierte mehrdimensionale Wellenpaket als ein Satz von zeitabhängigen Basisfunktionen dargestellt. Diese zeitabhängigen Basisfunktionen werden Einteilchenfunktionen (SPF) genannt. Die SPFs werden in der primitiven zeitunabhängigen Basis oder Gitter dargestellt. Das MCTDH-Verfahren kann als eine Zweilayerdarstellung angesehen werden: So bilden die Entwicklungskoeffizienten, die genutzt werden, um die korrelierte Wellenfunktion in dem Satz der SPF-Basis darzustellen, den oberen Layer der Darstellung. Die zeitabhängigen Entwicklungskoeffizienten, die die zeitabhängigen SPFs in der primitiven zeitunabhängigen Basis oder Gitter darstellt, bilden den unteren Layer. Die Bewegungsgleichungen, durch die gleichzeitig die optimale Entwicklungskoeffizienten für beide Layer 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, die, verglichen mit der primitiven Basis, signifikant kleiner gewählt werden. 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]

Das Moden-Kombinationsverfahren wurde von Meyer und seinen Mitarbeitern eingeführt^[48,67], das die Anzahl der Konfigurationen reduziert. Im ModenKombinationsverfah-

ren werden die „logische“ Koordinaten, 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 sind die MCTDH Rechnung statt der korrelierten Koordinaten durch die Partikel limitiert. So konnten Systeme mit 15 - 24 korrelierten Freiheitsgraden^[31,32,49,68] und System-Bad-Modelle^[69–71] durch das MCTDHModenKombinationsverfahren behandelt werden. Zwar konnte durch dieses Verfahren Grenz zu höherer Dimensionalität verwschoben 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 (ml)-MCTDH-Verfahren^[3] überwunden werden. Die SPFs des MCTDH-Verfahrens können ebenfalls als MCTDH-Wellenfunktion dargestellt werden. Im daraus resultierende Zweilayer-MCTDH wird eine Dreilayerdarstellung der Wellenfunktion genutzt.: Der obersten Layer wird durch die zeitabhängigen Entwicklungskoeffizienten gebildet. Die Wellenfunktion wird in der SPF-Basis des ersten Layers dargestellt, d.h. SPFs des einfachen MCTDHs. Der mittlere Layer wird durch die zeitabhängigen Entwicklungskoeffizienten gebildet, die SPFs des ersten Layers in der SPF-Basis des zweiten Layers darstellen. Die zweite Layer-SPF-Basis ist der zusätzliche Layer, der im ml-MCTDH-Verfahren hinzugekommen ist. Schließlich werden im unterste Layer die SPFs des zweite Layers in der primitiven zeitunabhängigen Basis oder Gitter dargestellt. Durch eine rekursive Anwendung der MCTDH-Verfahrens können weitere Layer hinzugefügt werden. Mit dem ml-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. So lange 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 nicht immer in dieser Form dargestellt werden. Durch die Verwendung einer spezifischen zeitabhängigen Quadratur, die die Matrixelemente allgemeiner Potentiale effizient auswertet, können auch Matrixelemente solcher *ab initio* Potentialenergieflächen effizient berechnet werden. Diese Vorgehensweise wird correlation discrete variable representation (CDVR) ^[72,73] genannt.

Das ursprüngliche Vorgehen für das CDVR ^[72] beruht auf einem zeitabhängiges DVR-Gitter, das einer SPF-Basis entspricht. Somit kann das Standard-CDVR weder für modenkombinierte MCTDH-Rechnungen noch für Berechnungen mit dem ML-MCTDH-Ansatz verwendet werden.

Allerdings konnte ein CDVR, das ohne ein direktes Produktgitter auskommt, in modenkombinierte MCTDH-Rechnungen verwendet werden. ^[74] 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 CDVR- Auswertung des Potentials ineffizient sind. ML-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 ml-MCTDH-Wellenfunktion

Zunächst werden die Wellenfunktionen der Standardmethode, des Zweilayer-MCTDHs und des modenkombinierten MCTDH betrachtet, um anschließend den Ansatz der ml-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:

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

Die zeitabhängigen Koeffizienten $A_{j_1, \dots, j_f}^1(t)$ beschreiben die Bewegung der Wellenpakete. Die Darstellung der Wellenfunktion in Gleichung 2.1 kann auch als Einfachlayerdarstellung angesehen werden und die Hochzahl 1 von $A_{j_1, \dots, j_f}^1(t)$ soll darauf hin-

weisen, dass $A_{j_1, \dots, j_f}^1(t)$ ein Entwicklungskoeffizient des ersten (und einzigen) Layers ist. Im MCTDH-Verfahren wird ein zusätzlicher Layer 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.

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

Anschließend werden diese SPFs innerhalb der zeitunabhängigen primitiven Basis dargestellt:

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

Gleichung 2.3 beinhaltet einen Satz zusätzlichen Entwicklungskoeffizienten, $A_{m;j}^{2;\kappa}(t)$, der die zeitabhängigen SPFs in der zeitunabhängigen primitiven Basis darstellt. Die hochgestellte Zahl z der Koeffizienten $A^z(t)$ bezieht sich auf die Layertiefen. In Gleichung 2.3 folgt aus $z = 2$, dass Gleichung 2.3 den zweiten Layer darstellt. Das hochgestellte κ und der Index m von $A_{m;j}^{2;\kappa}(t)$ beziehen sich auf die m -te SPF der κ -te Koordinate.

Zur Visualisierung der Layerstruktur des MCTDHs dienen die Diagramme für die unterschiedlichen Darstellungen der Wellenfunktionen in Abbildung 2.1.1. 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 ausgefüllten schwarzen Kreise gekennzeichnet. So kommt in Abbildung 2.1.1a ein Satz von Koeffizienten A_{j_1, \dots, j_7}^1 vor, der durch den einzigen schwarzen Punkt gekennzeichnet ist. Jede Linie, die von solchen Kreisen führen, entspricht einem tiefgestellten Index aus $A_{m;j}^{2;1}$ 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 $A_{m;j}^{2;1}$, der durch die SPFs $\phi_m^{1;1}(x_1, t)$ dargestellt ist, mit dem Koeffizienten A^1 über eine Linie, die für den m -ten Index steht, verbunden. Über eine weitere Linie, die den j -ten Index darstellt, ist $A_{m;j}^{2;1}$ mit dem Koordinatdeskriptor x_1 verbunden. In Abbildung 2.1.1a ist der Koeffizient A_{j_1, \dots, j_7}^1 durch sieben Linien mit den entsprechenden Indizes von j_1 bis j_7 direkt mit den sieben Koordinatdeskriptoren x_1, x_2, \dots, x_7 verbunden und in Abbildung 2.1.1b. ist A_{j_1, \dots, j_7}^1 mit den sieben Sätzen an A-Koeffizienten $A^{2;1}, A^{2;1}, \dots, A^{2;1}$ verbunden. Die

Wellenfunktion aus Abbildung 2.1.1a ist direkt in der zeitunabhängigen primitiven Basis dargestellt. In Abbildung 2.1.1b gibt es einen intermediären Layer an zeitabhängigen SPFs.

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

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

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

$$\Psi(q_1^1, q_2^1, \dots, q_d^1, t) = \sum_{j_1=1}^{n_1} \dots \sum_{j_d=1}^{n_d} A_{j_1, \dots, j_d}^1(t) \cdot \phi_{j_1}^{1;1}(q_1^1, t) \cdot \dots \cdot \phi_{j_d}^{1;d}(q_d^1, t) \quad (2.4)$$

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

In Abbildung 2.1.1c ist das entsprechende Diagramm der Wellenfunktion eines siebendimensionalen System, dessen Koordinaten in logischen Koordinaten zusammengefasst

wurden, dargestellt. 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 einfachtesten Fall kommen im ml-MCTDH zwei Layer von SPFs vor. Anstelle die SPFs aus Gleichung 2.4 in der primitiven zeitunabhängigen Basen 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 ml-MCTDH-Wellenfunktion. Der Ansatz der ml-MCTDH-Wellenfunktion ist gegeben durch

$$\Psi(q_1^1, q_2^1, \dots, q_d^1, t) = \sum_{j_1=1}^{n_1} \dots \sum_{j_d=1}^{n_d} A_{j_1, \dots, j_d}^1(t) \cdot \phi_{j_1}^{1;1}(q_1^1, t) \cdot \dots \cdot \phi_{j_d}^{1;d}(q_d^1, t) \quad (2.6)$$

gegeben.

$$\begin{aligned} \phi_m^{1;\kappa}(q_\kappa^1, t) &= \phi_m^{1;\kappa}(q_1^{2;\kappa}, \dots, q_{d_\kappa}^{2;\kappa}, t) \\ &= \sum_{j_1=1}^{n_{\kappa,1}} \dots \sum_{j_{d_\kappa}=1}^{n_{\kappa,d_\kappa}} A_{m;j_1, \dots, j_{d_\kappa}}^{2;\kappa}(t) \\ &\quad \times \phi_{j_1}^{2;\kappa,1}(q_1^{2;\kappa}, t) \cdot \dots \cdot \phi_{j_{d_\kappa}}^{2;\kappa,d_\kappa}(q_{d_\kappa}^{2;\kappa}, t) \end{aligned} \quad (2.7)$$

$$\begin{aligned} \phi_m^{2;\kappa,\lambda}(q_\lambda^{2;\kappa}, t) &= \sum_{j=1}^{N_\alpha} A_{m;j}^{3;\kappa,\lambda}(t) \mathcal{X}_j^{(\alpha)}(q_\lambda^{2;\kappa}) \\ &\quad \left(\text{mit } \alpha = \lambda + \sum_{i=1}^{\kappa-1} d_i \right) \end{aligned} \quad (2.8)$$

In Gleichung 2.7 ist $\phi_m^{2;\kappa,\lambda}$ mit der dazugehörigen Koordinate $q_\lambda^{2;\kappa}$ ein SPF des zweiten Layers. Die Hochzahlen 2 bezieht sich auf die Layertiefe, sodass der SPF zum zweiten Layer gehört, wobei κ und λ die dazugehörigen Koordinaten kennzeichnen. Die Entwicklungskoeffizienten $A_{m;j}^{3;\kappa,\lambda}(t)$ werden verwendet, um dieses SPF darzustellen und die Entwicklungskoeffizienten $A_{m;j_1, \dots, j_{d_\kappa}}^{2;\kappa}(t)$ definieren die Entwicklung der SPFs des ersten Layers, die in der SPF-Basis des zweiten Layers entwickelt werden. In Gleichung 2.3 definierten diese Entwicklungskoeffizienten die SPFs des ersten Layers in der primitiven zeitunabhängigen Basis. Abbildung 2.1.1d zeigt das entsprechende Diagramm für ein siebendimensionales System in ml-MCTDH-Darstellung.

Da die Gleichungen der ml-MCTDH-Wellenfunktion schnell unhandlich werden, ist es einfacher, statt der Gleichungen, die Wellenfunktionen wie in Abbildung 2.1.1 als Diagramm anzugeben. Da jedes Diagramm die Wellenfunktionen eindeutig definert, können

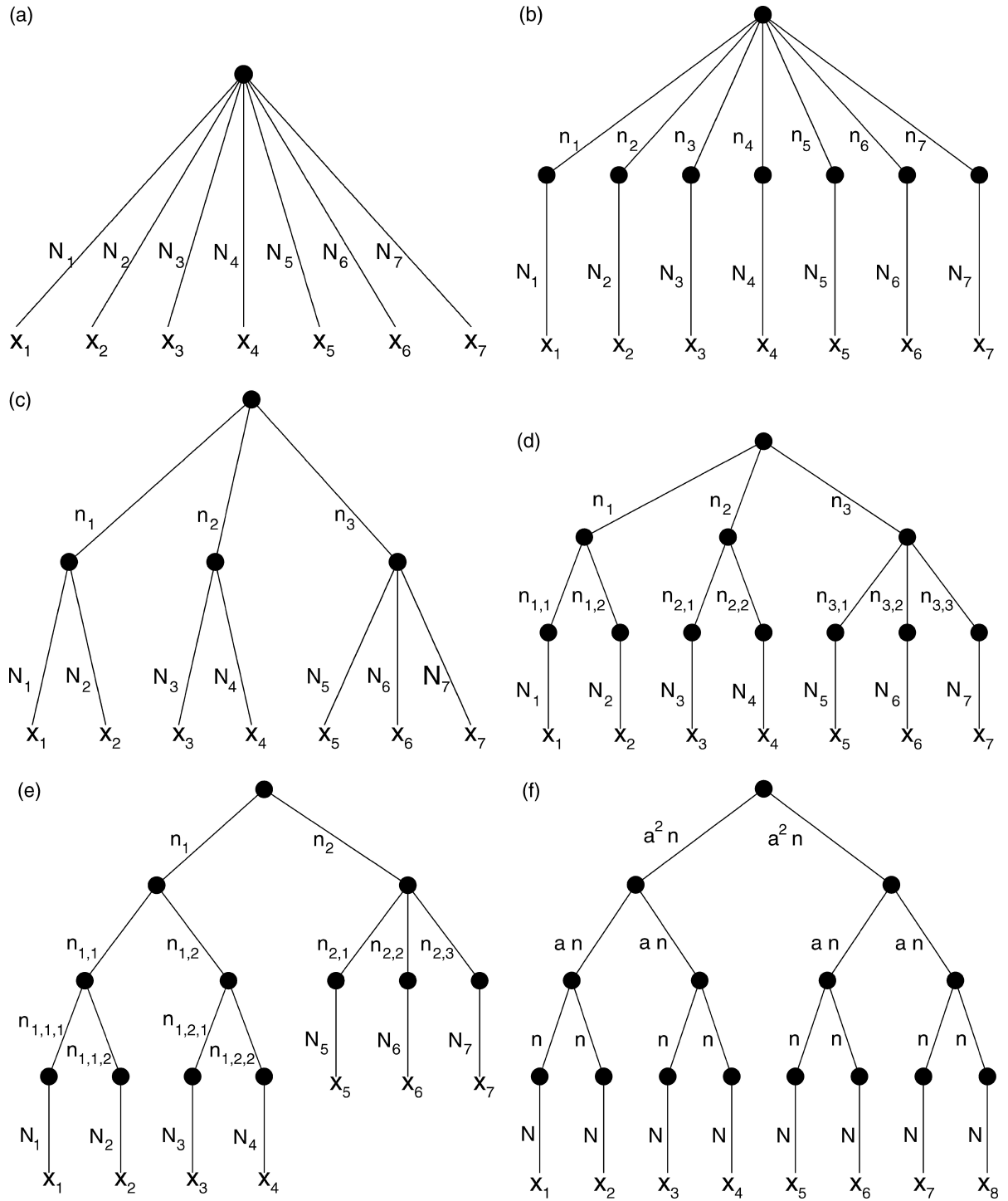


Abbildung 2.1.1: Unterschiedliche Darstellung von Wellenfunktionen eines siebendimensionalen Systems. Dargestellt sind: (a) Eine Darstellung eines Standardwellenpakets, (b) eine MCTDH-Wellenfunktion, (c) eine modenkombinierte MCTDH-Wellenfunktion, [(d)-(f)] eine ML-MCTDH-Wellenfunktion. Die Diagramme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

aus den Diagrammen die entsprechenden Gleichungen der Wellenfunktionen abgeleitet werden. Die Notation für die SPFs, A-Koeffizienten und (mehrdimensionalen) Koordinaten, die oben in den Gleichungen angegeben wurde, kann einfach für beliebige Multilayer erweitert werden.

Die ml-MCTDH-Darstellung kann mit einer beliebige Anzahl von Layern verallgemeinert werden. Daraus resultieren baumförmige Strukturen. Ein Beispiel dieser Strukturen ist in Abbildung 2.1.1e dargestellt. Die Anzahl der Layer für die jeweiligen Koordinaten kann bis zur primitiven Darstellung von Koordinate zu Koordinate variieren. So wird in Abbildung 2.1.1e das Baumdiagramm einer ml-MCTDH-Wellfunktion gezeigt, in der für die Koordinaten $x_1 - x_4$ drei MCTDH-Layer verwendet werden (d.h. insgesamt eine Vier-Layer-Darstellung) und für die Koordinaten $x_5 - x_7$ werden zwei MCTDH-Layer verwendet (d.h. eine Drei-Layer-Darstellung).

3 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 Features des MCTDH-Codes in Python verwendet werden kann. Es wäre denkbar, dass nützliche Python-Module wie NumPy (numerical Python), SciPy (Scientific Computing Tools for Python)^[62] und TensorFlow (machine learning)^[61] nahtlos mit dem MCTDH-Modul genutzt werden könnte.

Auch 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. Schließlich wäre auch das Implementieren der MCTDH-GUI als App auf Smartphones für die Benutzer hilfreich, von dem sie die laufenden Rechnungen überprüfen könnten.

4 Ergebnisse

4.1 Python-Interface für MCTDH

Es wurden eine Programmierschnittstelle (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 er ein Objekte von der Klasse *ControlParameter* initialisiert sein, bevor Methoden der Klasse *mctdhBasis* verwendet werden 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. In der Klasse *mctdhBasis* kann die Anzahl der Knoten des MCTDH-Baums aus dem Basisdatei ausgegeben werden. Knoteneigenschaften können über die Klasse *mctdhNode* ermittelt werden. Dabei stellt ein Objekte dieser Klasse eine Knoten dar. So kann festgestellt werden, ob ein Knoten den obersten Knoten darstellt oder zu den untersten Knoten gehört und mit wieviel weiteren Knoten er verbunden ist. 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 Demonstration 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 = {}
sumPerNode = {}
sumBottomNode = {}
sumTopNode = {}
remnantNodeList = []

def get_SPFs():
    SumTopNode = 0
    remnantNode = 0

    for i in range(maxNodes):
        node = basis.MCTDHnode(i)
        tdim = node.t_dim()
        nodes_spf[i] = tdim.GetnTensor()
    mode_spf = {i: basis.MCTDHnode(i).t_dim().active(0) for i in \
                range(maxNodes) if \
                basis.MCTDHnode(i).Bottomlayer() == True}

    for key in mode_spf:
        sumBottomNode[key] = mode_spf[key] + nodes_spf[key]
    BottomSum = sum([l_[1] for l_ in sumBottomNode.items()])

    for i in range(maxNodes):
        if basis.MCTDHnode(i).Toplayer() == True:
            children = basis.MCTDHnode(i).NChildren()
```



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

    return BottomSum + SumTopNode + 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 eine 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, werden alle SPFs, die zu einem Knoten führen multipliziert. Anschließend wird die Summe aller SPFs-Produkte gebildet. Hierfür werden zunächst die Summen

In den folgenden Python-Funktionen werden Objekte der Klasse *PhysCoor()*, *MctdhNode()* und *Tdim()* erzeugt.

In der ersten Funktion wird eine Liste zurückgegeben, die die untersten Knoten des eingelesenen MCTDH-Baums enthält. Dabei werden die Methoden *NmctdhNodes()* und *Bottomlayer()* verwendet, um die Anzahl sowie die Lage der Knoten ermitteln zu können. *Bottomlayer()* ist eine Methode von *MctdhNode()*. In der zweiten Funktion wird eine Liste der Moden des MCTDH-Baums mithilfe der Methode *mode()* von *PhysCoor()* zurückgegeben. Und die letzte Funktion generiert eine Liste aus SPFs, die mithilfe der Methode *GetnTensor()* von *Tdim()*. Die Objekte *node*, *phys* und *tdim* hätten auch durch

```
node = mctdh.MctdhNode()
phys = mctdh.PhysCoor()
tdim = mctdh.Tdim()
```

initialisiert werden können.

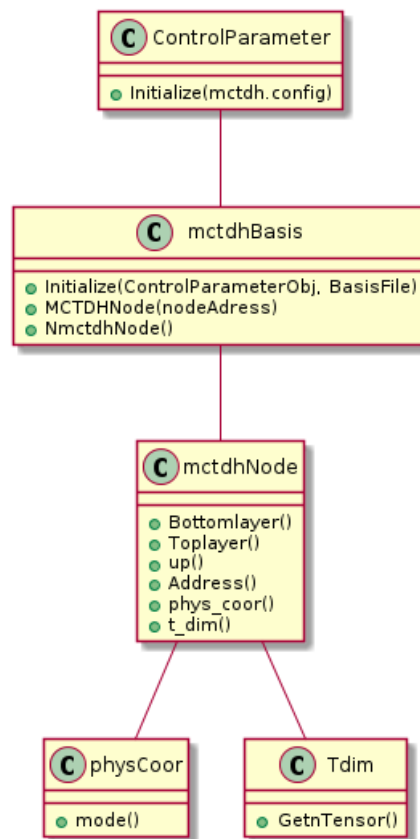


Abbildung 4.1.1: Alle Klassen, die in Cython erstellt wurden, sind mit einem „C“ gekennzeichnet. Die jeweiligen Klassenmethoden sind mit einem grünen Punkt gekennzeichnet.

4.2 Graphische Benutzeroberfläche für MCTDH

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.^[75]

In Abbildung 4.2.1 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 wurden, 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. Beispielsweise 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 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.^[75]

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, Checkboxes oder Elemente innerhalb eines Fensters emittieren Signale aus, die Aktionen des Benutzers zugeordnet werden können. Aktionen können das Einmal- oder Doppelklicken, das Bewegen des Mauszeigers oder das Betätigen 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 Datenmodell von der Darstellung der Daten trennt. Ein Datenmodell 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 Datenmodell und der graphischen Darstellung der Daten beruht auf dem Model-View-Controller (MVC) Paradigma.^[76]

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).^[77]

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 verschiedenen Ansichten dargestellt werden. Die Implementierung neuer Darstellungsarten ändert nicht die darunterliegende Datenstruktur.^[76] 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 Datenmodell 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.

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 Harreveld, G. Nyman, and U. Manthe, J. Chem. Phys. **126**, 084303 (2007).
- [40] G. Nyman, R. van Harreveld, 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] M. Ehara, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **105**, 8865 (1996).
- [68] C. Cattarius, G. A. Worth, H.-D. Meyer, and L. S. Cederbaum, J. Chem. Phys. **115**, 2088 (2001).
- [69] H. Wang, J. Chem. Phys. **113**, 9948 (2000).
- [70] H. Wang, M. Thoss, and W. Miller, J. Chem. Phys. **115**, 2979 (2001).

- [71] M. Nest and H.-D. Meyer, J. Chem. Phys. **119**, 24 (2003).
- [72] U. Manthe, J. Chem. Phys. **105**, 6989 (1996).
- [73] R. van Harreveld und U. Manthe, J. Chem. Phys. **121**, 5623 (2004).
- [74] R. van Harreveld and U. Manthe, J. Chem. Phys. **123**, 064106 (2005).
- [75] M. Summerfield, *Rapid GUI Programming with Python and Qt* (Prentice Hall, 2008).
- [76] *Model/view programming* (2018), URL <http://doc.qt.io/qt-5/model-view-programming.html>.
- [77] G. Krasner and S. Pope, Tech. Rep., ParcPlace Systems, Mountain View, Calif. (2002).