ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Writing Hagedorn wavepackets in C++ with HDF5 Interface

Bachelor Thesis

Florian Frei

August 25, 2016

Advisors: Prof. Dr. Ralf Hiptmair, Dr. Vasilie Grădinaru
Seminar of Applied Mathematics, ETH Zürich

Abstract

It is of general interest in physics/chemistry to find viable algorithms to solve the time dependent Schrödinger equation. A means to an end are *Hagedorn* wavepackets for a semi-classical approach. With this as basis a python implementation was created. This implementation can be used to simulate different kind of starting conditions which can result in a lot of data. To efficiently store this data it is desirable to use binary formats, which can be easily compressed if needed. A well-known data binary format is HDF which stands for hierarchical data format. To further improve simulation speed the python implementation was used as a basis for a C++ implementation. This thesis explains the intrinsic functionality of using the HDF5 library interface in C++. This was done in such a way that the data from the C++ implementation has the same hierarchy as the python implementation. Also for further use a test was implemented with the well-known GoogleTest framework to easily compare possible data between these two implementations.

Contents

Chapter 1

Introduction

1.1 Motivation

To test an algorithm or approach such as semiclassical wavepackets [1] for solving the time dependent Schrödinger equation a python implementation [3] is at most times sufficient. In case longer simulation times and/or solving higher dimensional problems is demanded the computation time becomes an important factor. As such there is a need for a time efficient implementation which is only doable in languages such as C or C++. A starting implementation in C++ for scalar wavepackets is already done [2] but further improvement is desired. One aspect which has to be taken into account is the storage format of the data but also the speed of these io-operations. As storage format the well-know HDF(hierarchical data format) is most suitable for our case given that it is a binary file format with sufficient io speed. The current form to store data is dependent on a external source [4], which is sufficient for small projects where data is supervised by the user, which uses HDF5 format but doesn't fully use its capabilities. Furthermore the chosen hierarchy is also not compatible with the dual python implementation. In this project a fully compatible hierarchy is implemented in C++ without the usage of an external source. Also for testing simulations under both implementations a data test was implemented using the well-know Google testing framework.

1.2 Background

$$i\hbar \frac{\partial}{\partial t} |\varphi\rangle = H |\varphi\rangle \tag{1.1}$$

Chapter 2

HDF5 C++ Interface

2.1 Overview

The acronym HDF stands for hierarchical data format meaning this binary format is allowing to structure the internal objects after the users demand. This structure is quite similar to a file system where data is ordered with folders and sub-folders. For more detailed information see the official C [5] and C++ [6] documentation. The reason why also the C documentation is very important is based on the fact that the C++ implementation is mostly a nice wrapper based on the C implementation. For the exact dependence see table 2.1. To use the C++ language features to its most capabilities in-

HDF5 C APIs	C++ Classes
Attribute Interface (H5A)	Attribute
Datasets Interface (H5D)	DataSet
Error Interface (H5E)	Exception
File Interface (H5F)	H5File
Group Interface(H5G)	Group
Identifier Interface (H5I)	IdComponent
Property List Interface (H5P)	PropList and subclasses
Dataspace Interface (H5S)	DataSpace
Datatype Interface (H5T)	DataType and subclasses

Figure 2.1: Table of correspondence between C and C++

heritance is used. This allows reuse of functions, objects and properties over many hierarchy layers. This enforces a strict dependence when sharing or creating objects. To have an overview on the hierarchy look at figure 2.2.

As stated in the background this project is about the simulation of *Hagedorn* wavepackets over a fixed time horizon. To keep track and possibly repro-

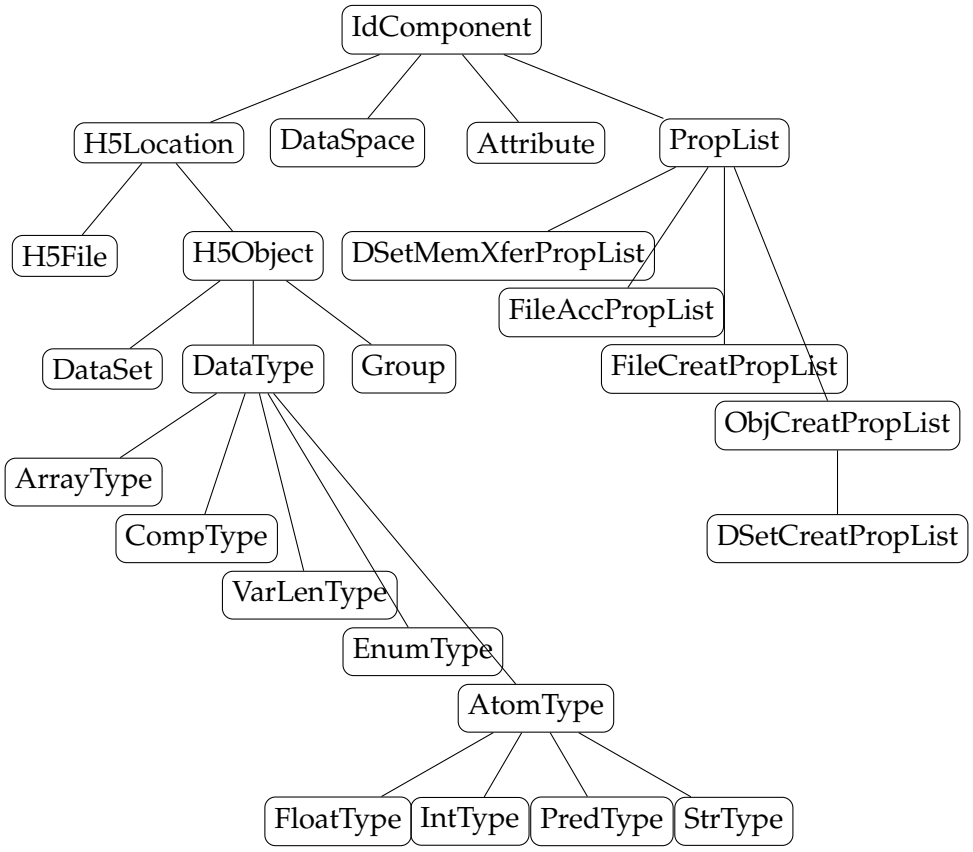

Figure 2.2: Depiction of derivation hierarchy

duce the results the wavepackets have to be saved in every time step Δt of the simulation. This has to be achieved with the classes shown in figure 2.2. To understand the functionality of these classes they will be explained in the following sections.

2.2 Internal types and states

An interface in general has supported objects and functions. These functions also have preconditions and postconditions whereas objects have valid states. In case of the HDF5 library when these are not fulfilled an exception is thrown which could abort or interrupt the program execution. Therefore it is important to always use valid objects and function calls. There are two noteworthy types which are also used the most. These are `hsize_t` and `H5std_string`. Variable of type `hsize_t` represent native multiple-precision integer. This type substitutes the C++ internal `int` data type. The `H5std_string` type is just an alias for the `std::string` data type from the standard library. Internal valid states are represented in only uppercase letters and with the C

prefix as in table 2.1. For example a valid property list state is `H5P_DEFAULT`.

2.3 H5File

As the name already suggest this class is used to manage the binary file object. When a *H5File* is default constructed it also allocates a default *Group* root named "/". To construct a *H5File* a minimal number of two arguments is needed. The two additional optional arguments are a *FileCreatPropList* and a *FileAccPropList* which would allow further specification. These would be creation and access properties as suggested by the names. In case of this project they are not needed and the `H5P_DEFAULT` is sufficient. For the first mandatory argument, which is the filename, a `H5std_string` or as discussed before a string required. The second mandatory argument defines the type of creation which at default has two valid values. These possible values are *H5F_ACC_TRUNC* and *H5F_ACC_EXCL* which are mutually exclusive. The former truncates the file meaning if it already exists erase all data previously stored in the file. The latter fails if the file already exists under the specified name which ends in an exception. It is advised to work with *H5F_ACC_TRUNC* because it is simpler and thus errors will less frequently occur.

2.4 Group

As previously mentioned it is a hierarchical data format. The hierarchy gets implement through the usage of *Groups*. In case of a *Hagedorn* wavepackets and its corresponding energies the structure in figure 2.3 is preferred as it corresponds to the structure of the python generated data.

To create a *Group* only a `string` argument is required which acts as name but more importantly also as path similar to figure 2.3. As previously indicated a *H5File* has root *Group* "/" by default after allocation. To accomplish the structure as in figure 2.3 the path is included in the name. For instance to have a *Group* "wavepacket" after the root "/" and a *Group* "Pi" after "wavepacket" the first name will be modified to "/wavepacket" and the second to "/wavepacket/Pi". This implies that every intermediate node in figure 2.3 will be a *Group*. The leafs however will be *DataSets* which will be explained in the next section 2.5.

2.5 DataSet

The constructor of the *DataSet* class demands four arguments with individual type *string*, *DataType*, *DataSpace* and *DSetCreatPropList*. The first argument analogous to *Group* is the name with its path included. For example

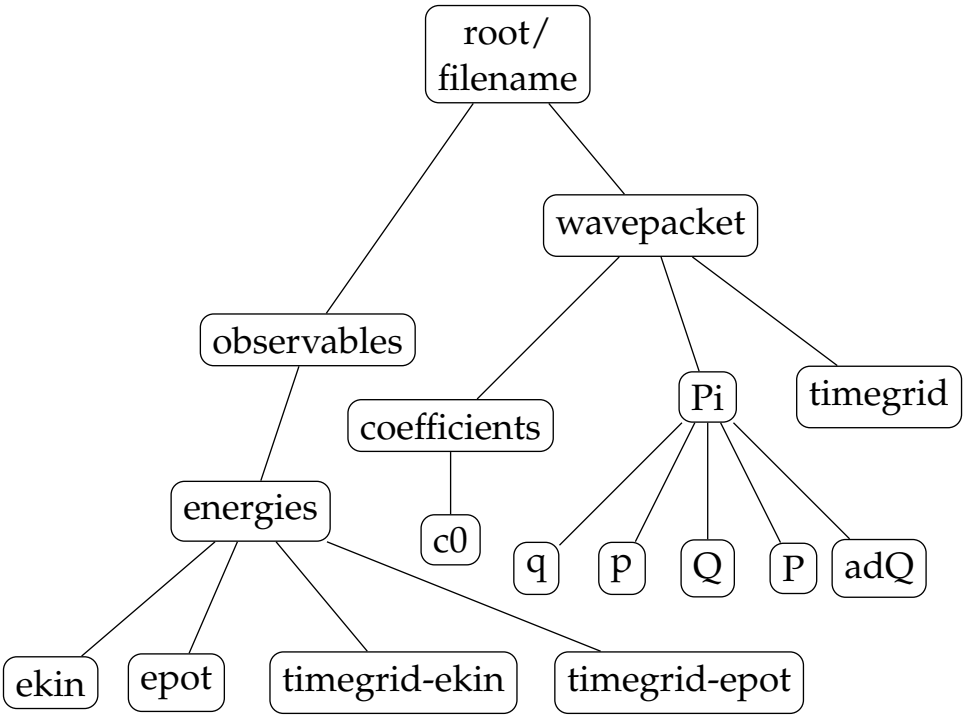

Figure 2.3: Depiction of desired internal structure of a H5File

the *DataSet* "ekin" has the complete name "/observables/energies/ekin". As the name suggest the second and third argument specifies the type and space of the data. Lastly the forth argument defines the properties at creation. This incorporates which data layout is chosen. The *DataSet* has three types of layouts to store raw data. These are `H5D_COMPACT`, `H5D_CONTIGUOUS` and `H5D_CHUNKED`. Figure 2.4 should illustrate the inner workings of these layouts.

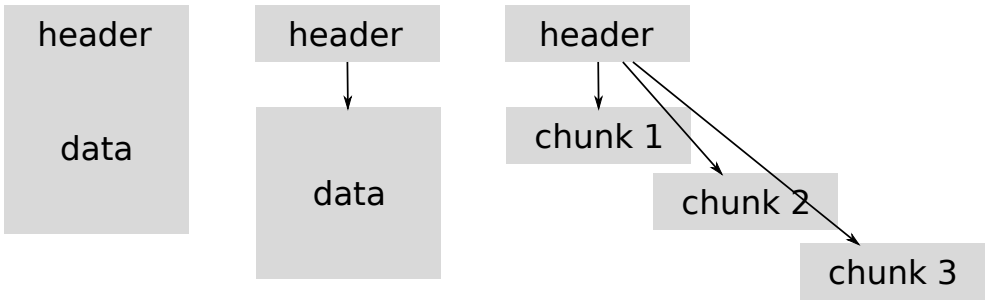

Figure 2.4: The three data layouts in *DataSet*

If the data is not to large we can store it as `H5D_COMPACT` meaning the data follows right after the header shown on the left. In case the data is larger

but still somehow constant we use the H5D_CONTIGUOUS layout in the middle meaning the header points to the data block which is stored contiguously in memory. The last case on the right is used if the size of the data is unknown. We store in the header the locations of all chunks meaning to add new data a new chunk will be appended in the header.

2.6 DataType

The language C++ itself supports data types such as int, double, char etc. but these cannot be used directly in a binary data format. The reason for this

2.7 DataSpace

2.8 PropList

2.8.1 DSetCreatePropList

2.9 Attribute

Chapter 3

Writer Template

Chapter 4

Data Test

4.1 Introduction to GoogleTest

4.2 The Main C++ File

Chapter 5

Conclusion

Appendix A

Dummy Appendix

You can defer lengthy calculations that would otherwise only interrupt the flow of your thesis to an appendix.

Bibliography

[1] R. Bourquin. Algorithms for non-adiabatic transitions with one-dimensional wavepackets. 2010. http://www.sam.math.ethz.ch/~raoulb/research/bachelor_thesis/tex/main.pdf.

[2] R. Bourquin, M. Bösch, L. Miserez, and B. Vartok. libwaveblocks: C++ library for simulations with semiclassical wavepackets. https://github.com/WaveBlocks/libwaveblocks, 2015, 2016.

[3] R. Bourquin and V. Gradinaru. WaveBlocks: Reusable building blocks for simulations with semiclassical wavepackets. https://github.com/WaveBlocks/WaveBlocksND, 2010 - 2016.

[4] James R. Garrison. eigen3-hdf5. https://github.com/garrison/eigen3-hdf5, 2013.

[5] HDF Group. HDF5 C Documentation. https://www.hdfgroup.org/HDF5/doc1.6/RM_H5Front.html, 2001.

[6] HDF Group. HDF5 C++ Documentation. https://www.hdfgroup.org/HDF5/doc/cpplus_RM/index.html, 2001.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.