



MSc in Physics

# Wave Equations without Coordinates

Developing computational and mathematical methods for calculating wave equations without coordinates on two-dimensional manifolds arising from fullerene molecules.

Simon Krarup Steensen

Supervised by Assoc. Prof., Ph.D. James E. Avery

November 3, 2020



**Simon Krarup Steensen**

*Wave Equations without Coordinates*

MSc in Physics, November 3, 2020

Supervisor: Assoc. Prof., Ph.D. James E. Avery

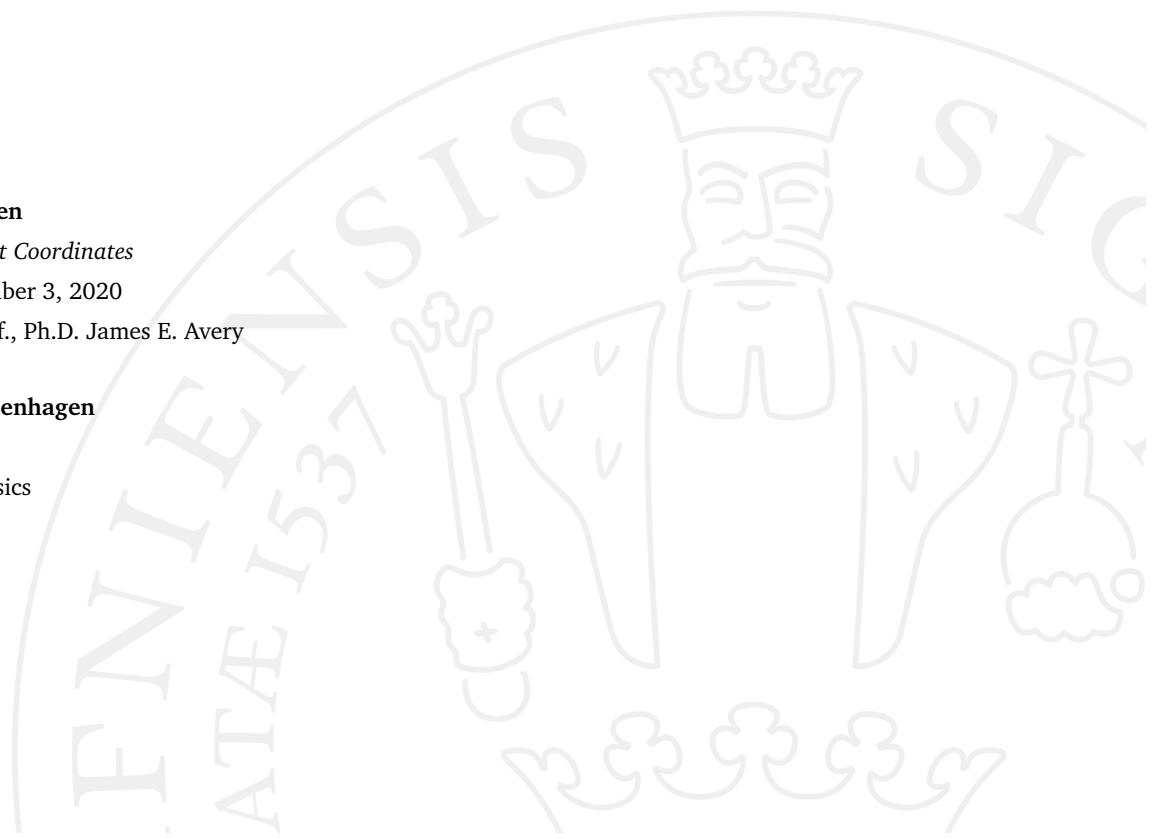
**The University of Copenhagen**

*The Niels Bohr Institute*

Masters Degree in Physics

Blegdamsvej 17

2100 Copenhagen Ø



# Abstract

Closed polyhedral molecules consist of atoms which form connecting polygons creating a 2-dimensional non-euclidean surface e.g. the carbon allotrope of the fullerenes. The geometrical properties of the fullerenes make them well suited for systematic analysis. The question of interest to this work is, if it is possible to approximate an electronic density along these surface manifolds without ever needing the 3-dimensional embedding. The hope that this is in fact possible builds on the assumption that the electron mobility on the graphene like surface dominates electronic interaction through the hollow volume holds.

This would be of interest since a single isomer DFT analysis can take days to weeks using a supercomputer, with the number of possible  $n$ -atom isomers growing as  $\mathcal{O}(n^9)$ . The geometrical properties of fullerenes however allow for fast computations in microseconds per isomer of fullerene bond graphs. This then allows for rapid construction of an isomer's 2-dimensional manifold surface. A 2-dimensional density functional theory approach to this manifold, without ever needing quantum mechanical geometrical optimization, would theoretically reduce computations drastically.

The basis for this thesis is in approaching the problem in a purely 2-dimensional setting of surface manifolds arising from the fullerene polygons. The hope is that a 2-dimensional DFT can produce densities that encapsulates key features of the true fullerene electronic densities. This thesis will by developing mathematical and computational approaches create a PDE solver for the non-euclidean manifold surfaces. The PDE solver is a finite element method in which a discrete Laplace-Beltrami operator is introduced to account for the curved space surfaces. This can be used to compute solutions to PDEs without any reference to global coordinates of the fullerene 3-dimensional embedding. The only geometrical input needed is the sparse adjacency matrix of the fullerene's cubic graph representation. This is then applied to the heat equation to help validate the implementations, as well as Poisson's equation and the Kohn-Sham equations. A rudimentary 2-dimensional DFT method developed then solve Poisson's equation and the Kohn-Sham equations in an iterative manner using the solver. The DFT has implemented a local density approximation accounting for the exchange correlation potential, but no potential arising from nuclear attraction on the surfaces.

Results for the PDEs are presented on various fullerene surfaces to validate the implemented PDE solver. Results for the preliminary DFT are of electronic densities confined to the 2-dimensional surfaces with no nuclear attraction present. Electronic densities converging to a steady distribution in the DFT are presented for the  $C_{20}$ - $I_h$ , a  $C_{60}$  nanotube and a  $C_{120}$ - $D_6$  surfaces. With the electronic densities confined to the non-euclidean surfaces presented considerations about the outlook and the challenges of future research will be discussed.

# Acknowledgements

I would first of all like to thank James Emil Avery for his countless hours of indispensable supervising during the project. Being a part of the fullerene research project has been an incredible instructive process that I feel fortunate to have participated in. The interplay of many, at the time, unfamiliar theoretical concepts and especially the future big picture plans is something I still find intriguing.

I would also like to thank the people within the fullerene group for the comradery in our office. This meant that a sparring mate was always close by to discuss frustrating computational problems or tough theoretical concepts. Even when the society was in lockdown, James opened his garden for group meetings with social distancing, which allowed for an understanding of each others project, which aided in seeing the present and future interplay between the projects.

Lastly I want to thank my family and friends for their general support. While an interesting subject is important to enjoy the thesis process their support can not be understated.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Theory</b>	<b>5</b>
2.1	Fullerenes . . . . .	5
2.1.1	Gaussian Curvature . . . . .	5
2.1.2	Graph Theory . . . . .	7
2.1.3	2-Dimensional Unfolding of Dual Representation . . . . .	9
2.1.4	Generalization to Fulleroids . . . . .	9
2.2	Finite Element Method . . . . .	10
2.2.1	FEM Overview . . . . .	10
2.2.2	Variational Formulation . . . . .	11
2.2.3	Ritz-Galerkin Approximation . . . . .	12
2.2.4	Discretization of Domains . . . . .	13
2.3	The Quantum Mechanical Many-Body Problem . . . . .	16
2.3.1	Born-Oppenheimer Approximation . . . . .	17
2.3.2	Density Functional Theory . . . . .	18
<b>3</b>	<b>Method</b>	<b>23</b>
3.1	FEM Implementation . . . . .	23
3.1.1	Triangulation of Fullerenes . . . . .	24
3.1.2	FEM Matrices and Vector Assembly . . . . .	25
3.1.3	The Reference Element . . . . .	26
3.1.4	Integral Evaluation with Gaussian Quadrature . . . . .	29
3.1.5	Stiffness Matrix in Curved Space . . . . .	29
3.2	FEM Modules . . . . .	32
3.2.1	Basis Modules . . . . .	32
3.2.2	UnitCell Module . . . . .	33
3.2.3	FEM_Assembly Module . . . . .	36
3.3	Constructing a 2-Dimensional DFT with FEM Software . . . . .	39
3.3.1	Exchange-Correlation Potential . . . . .	39
3.3.2	Local Density Approximations . . . . .	39
3.3.3	The Self-Consistent-Field Loop . . . . .	41
3.3.4	Computing the SCF loop . . . . .	42
3.4	Solving the Heat Equation . . . . .	45
3.5	Visualization . . . . .	46
3.5.1	Dual Unfolding . . . . .	46
3.5.2	2-Dimensional Visualization . . . . .	47

3.5.3	Mesh Refinement . . . . .	48
3.6	Constructing a Torus Shaped Fulleroid . . . . .	49
<b>4</b>	<b>Results and Discussion</b>	<b>50</b>
4.1	Simulating the Heat Equation . . . . .	51
4.1.1	Simulations on the $C_{20}$ - $I_h$ surface . . . . .	51
4.1.2	Heat equation on a $C_{60}$ -Nanotube Surface . . . . .	55
4.2	A Rudimentary 2-Dimensional DFT . . . . .	58
4.2.1	Solutions for the Hartree potential . . . . .	58
4.2.2	Kohn-Sham Orbitals . . . . .	64
4.2.3	SCF Loop . . . . .	69
<b>5</b>	<b>Outlook</b>	<b>76</b>
5.1	FEM Related Challenges . . . . .	76
5.2	DFT Related Challenges . . . . .	76
<b>6</b>	<b>Conclusion</b>	<b>78</b>
	<b>Appendices</b>	<b>80</b>
<b>A</b>	<b>Appendix</b>	<b>81</b>
A.1	Convergence of C20 Simulations . . . . .	81
<b>B</b>	<b>Appendix</b>	<b>82</b>
B.1	Directory Overview of the Software . . . . .	82
<b>A</b>	<b>Bibliography</b>	<b>83</b>

# Introduction

Fullerenes are closed-surface carbon molecules formed by pentagon and hexagon faces. The very first fullerene experimentally discovered in 1984 was the  $C_{60-I_h}$  Buckyball[1], which consist of 12 pentagons and 20 hexagons, with all pentagons being separated by exactly one hexagon to each side. The familiar structure of the modern-day football but was named Buckminsterfullerene, in honour of the, at the time, recently deceased architect Richard Buckminster Fuller, as the new molecule formed a polyhedron by the same mathematical rules as Fuller used to construct his experimental architectural pieces named geodesic domes<sup>1</sup>. The Buckminsterfullerene has since its discovery been found to have widespread applications in various fields with uses ranging from solar cells[2] to immunology as asthma medicine[3], and have even been theorized to have the ability of packing hydrogen at near-metallic densities[4]. The discovery of the  $C_{60-I_h}$  fullerene was a landmark in chemistry and earned its discoverers Kroto, Curl and Smalley the Noble Prize in chemistry in 1996. It has later been noticed to be the most common fullerene appearing in nature and have even been detected in space by the Spitzer space telescope[5] along with a few other fullerenes of similar size.

Fullerenes always have exactly 12 pentagons by Euler's polyhedral formula. It is the placement of these pentagons and the number of hexagons that fully determine the geometrical structure. The bonds which outline the polygons are  $sp^2$  hybridised and each carbon atom will therefore form three bonds to its nearest neighbours. As the number of atoms increase, the number of non-isomorphic<sup>2</sup> fullerenes grow as  $\mathcal{O}(n^9)$  [6]. For the smallest fullerene  $C_{20-I_h}$  which consist of only 12 pentagons there exists only 1 structure, 1812 exists for  $C_{60}$ , 285,914 for  $C_{100}$ , 132,247,999,328 for  $C_{400}$  etc.[7]. This incomprehensible number of different molecules leaves us optimistic, since the few fullerenes that have been synthesized have as mentioned found uses in various technologies. The large number of possible untested structures therefore leaves a promising outlook of their capabilities. But while the many geometrical combinations of possible fullerenes seem exciting, it leaves quite a few challenges.

One challenge is how one can actually produce the molecules. Producing the Buckminsterfuller is commonly done through laser ablation or resistive heating of graphite, which is only possible due to the high stability of the molecule. These methods are non-selective and can not be applied as a general technique of producing various isomers, thus we are in need of an alternative approach. A possible solution to this was presented in [8] where  $C_{60}$  molecules were synthesized from an auto-assembling planar precursor. The precursor in question contained all 60 carbon atoms and 75 of the total 90 carbon bonds. Also present in the precursor molecule were chlorine atoms at specific positions which when exposed to vacuum pyrolysis assembles into the spherical Buckminsterfuller. This has later been used to produce other nearly spherical fullerenes such as  $C_{78}$ [9] and  $C_{84}$ [10]. Scaling this method should be possible in theory, leaving the open ended question: What fullerenes are of interest to actually synthesize?

---

<sup>1</sup>His geodesic domes actually corresponds to a dual representation of fullerenes which consist of equilateral triangles.

<sup>2</sup>i.e geometrically unique

If we wish to compute whether or not a specific fullerene isomer have certain desired physical/chemical properties, that will be worth the resources to synthesize, we turn to ab initio quantum chemistry calculations. Due to the large number of electrons in fullerenes the highest level of ab initio theory we can feasible use is Density Functional Theory(DFT). A detailed DFT calculation though possible, still take anywhere from days to months in a supercomputer for a single isomer dependent on the size of the molecule, basis set and complexity of the used DFT method. A single  $C_{400}$  isomer would realistically take several months on a super computer, which is just one unique structure with over a hundred billions molecules of the same size. Another approach is therefore needed if we wish to search through isomers spaces to find structures with desired molecular properties.

While the aforementioned detailed DFT calculation depend on computing the 3-dimensional geometry of the isomer in question, a new alternative speculative approach is suggested by J. Avery [11] which is subject to research in the eScience Fullerene group. Describing a fullerene purely by its cubic graph representation will allow to fully describe the system in a 2-dimensional manner without ever referring to the 3-dimensional embedding. The idea is then to develop a 2-dimensional DFT method in the hope that it will approximate well to the true 3-dimensional electronic density. The justification for this is that the extreme electron mobility along the graphene like surfaces dominate over electronic interactions through the hollow volume. The validity of this assumption obviously depend on the size and shape of the fullerene in question.

A DFT implementation however include the task of solving partial differential equations which we wish to do on the 2-dimensional surfaces. For this the finite element method (FEM), which usually find it's applicability in analysis in engineering arts seem like a fitting approach. This is due to the fact that a FEM can be tailored for this problem by solving partial differential equations purely based on the cubic graph representation. The partial differential equations relevant to a DFT implementation such as the Kohn-Sham equations can therefore be constructed without any use of global coordinates, hence the title of the work at hand.

## The Work at Hand

This thesis will lay the groundwork for the stated challenges by developing a computational partial differential equation solver for the relevant equations within DFT. The constructed solver will be based on the FEM with an integrated curved space discrete Laplace-Beltrami operator. Once this is in place a rudimentary 2-dimensional DFT is developed. For this a local density approximation is applied to account for the exchange-correlation potential. The DFT developed in this thesis deals with simulations on the manifolds with no positively charged nuclei present. This does obviously not represent the nature of an electronic density on fullerenes, but it will lay the groundwork for further research.

## Future Work

Finite element methods are not commonplace in DFT analysis<sup>3</sup>. This is mainly due to their inability of representing the exponential atomic cusps of wavefunctions and densities at nuclear centers. Here pseudopotentials may aid in ever having to represent exponential behaviour. In a DFT where we would

---

<sup>3</sup>Same goes for finite difference methods.



like to find electronic densities for the valence electrons pseudopotentials aim to represent the effective potential arising due to nuclear attraction and electron repulsion of the inner shell electrons.

There are plans within the Fullerene group to research mapping the graphene solution onto the fullerene as an underlying potential. The hope is within the group that a general potential arising from the graphene solution can be mapped onto the fullerene as an underlying potential. Having an underlying solution to graphene will hopefully not only avoid cusps we can not represent, but also to investigate where the deviation differs from graphene, which is of great interest.

In fullerene regions of special interest, in which the 3-dimensional electronic interaction are dire will mainly be areas associated with pentagons since they create the curvature in the fullerene. It is mainly here the behaviour is expected to deviate from graphene. In these special points the fullerene research group hope an extension to a simplicial complex with a 3-dimensional representation around such a region is possible. Several challenges subject to present or future research and further explanations to the above problems will be given section 5.

Keeping in mind that a large class of molecular properties can in general be found through derivatives of the energy in the molecule  $E^4$  i.e.

$$Property \propto \frac{\partial^{n_F+n_B+n_I+n_R} E}{\partial \mathbf{F}^{n_F} \partial \mathbf{B}^{n_B} \partial \mathbf{I}^{n_I} \partial \mathbf{R}^{n_R}}, \quad (1.0.1)$$

with the derivatives being with respect to the electric-field  $\mathbf{F}$ , magnetic-field  $\mathbf{B}$ , nuclear spin  $\mathbf{I}$  and the molecular geometry  $\mathbf{R}$ . If the 2-dimensional speculative research is successful in approximating true density appropriately it would allow us to search through large isomer spaces and pick out isomers of desired approximate properties. These isomers can then be investigated further by producing their 3-dimensional geometry from their graph using force field optimization. This would allow for 3-dimensional projection of the surface densities. If such a projection approximate well enough to physical reality we could in turn derive molecular properties which need an 3-dimensional embedding.

## Thesis Content

This thesis encapsulates the entirety of my work in the eScience group at the Niels Bohr Institute at the University of Copenhagen. The research at hand dealing with computational, physical and mathematical aspects of fullerenes is led by associate professor James Emil Avery. An overview of the research currently being conducted as well as future tasks can be found at <https://www.nbi.dk/avery/CARMA/index.html>. It is currently funded through Avery's VILLUM Experiment project 00023321 "FoldingCarbon: A Calculus for Molecular Origami".

The thesis will at the beginning of each chapter give a short introduction to the following sections and outline how they fit together. It will through the theory in chapter 2 and the code constructed in chapter 3 produce solutions to differential equations on fullerene surfaces. The solutions will be presented in chapter 4 by visualizing fullerenes surfaces as 2-dimensional unfoldings. Besides the differential equations relevant for DFT, simulations of the heat equations on the surfaces will be computed to help validate the FEM implementation.

To show the adaptability and flexibility of the computational approach, an extension of the code to also analyze fulleroid structures will be given as well. Fulleroids allow for heptagons, octagons etc. to be

---

<sup>4</sup>which is obtained in a DFT.

included in fullerenes. This creates negative curvature and allow for countless new structures including closed structure of genus  $g \neq 0$ . Among one possibility is the torus with  $g = 1$ , on which the Kohn-Sham equations are solved to show the implementation's general applicability.

## Software

The software written and used for the thesis can be found at Github repository

<https://github.com/SKS94/Fullerene-Thesis>

Here various computational implementation relevant to the stated challenges are located. The different tasks solved by the written software are in broad strokes

- FEM modules using only the fullerenes cubic graph connectivity information  
<https://github.com/SKS94/Fullerene-Thesis/tree/FEM>
- DFT functions including computations of the self-consistent field loop  
<https://github.com/SKS94/Fullerene-Thesis>
- A visualization module for displaying solutions on 2-dimensional unfoldings  
<https://github.com/SKS94/Fullerene-Thesis/tree/Plotting>
- Scripts creating animations of the heat equation simulations on fullerene manifolds  
<https://github.com/SKS94/Fullerene-Thesis/tree/Heat>

A directory of the implemented modules can be found in appendix B. Key functions will be presented in chapter 3, highlighting the important aspects of the implementations. Before we dive into this work head first it only seems fitting with a slightly cheesy inspirational quote from the fullerenes eponym

*"You can never learn less, you can only learn more"*

*R. Buckminster Fuller*

# Theory

The following section aims to cover the most essential theoretical concepts on which this work is build. Starting with a general introduction to fullerenes in section 2.1 with focus on geometrical aspects that aid in describing them, such as Gaussian curvature and graph theory. This section will end with a generalization of fullerenes called fulleroids. Section 2.2 will follow with a more general introduction to key concepts in finite elements methods. This will explain the approach used to solve partial differential equations on manifolds. Lastly a section 2.3 explain the general ideas and concepts on which density functional theory stand since this is a vital premise for the work at hand.

## 2.1 Fullerenes

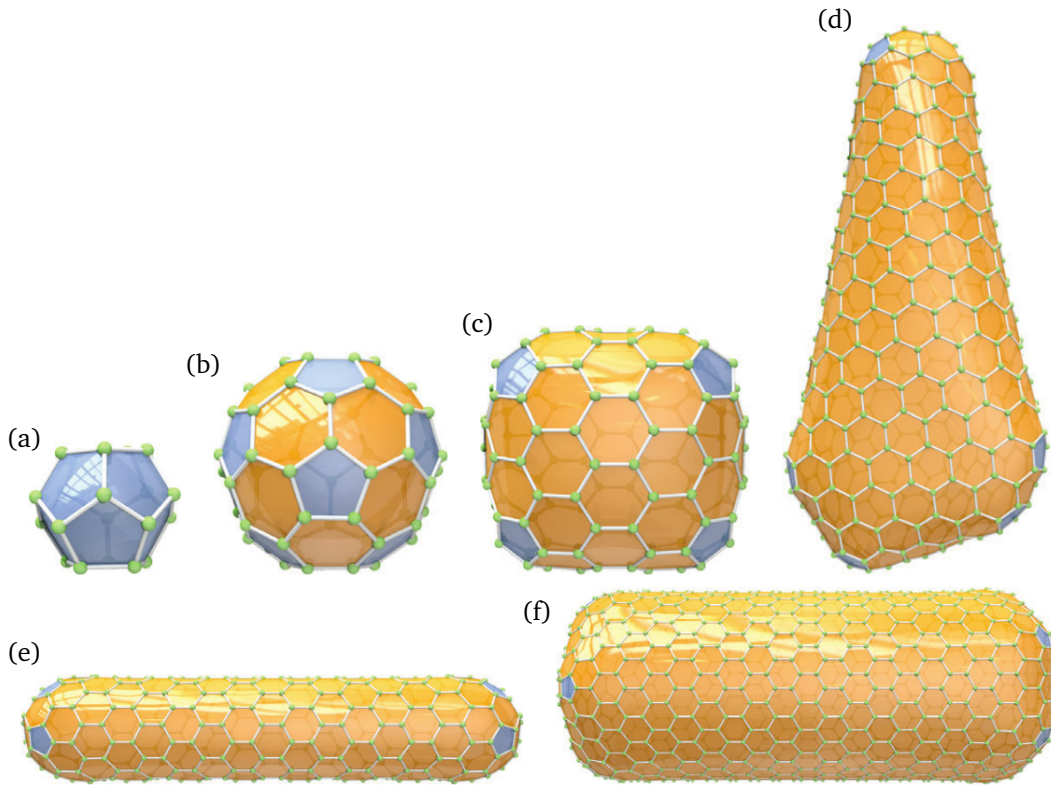
The allotropes of carbon many of us are familiar with include graphite and diamonds, but carbon can form many exotic materials and molecules including graphene, nanotubes, cyclocarbon etc. Graphene is a carbon allotrope of only one atoms thickness consisting of connected hexagons forming a honeycomb lattice. The material holds several interesting properties and is the subject to ongoing research. The alloptrope the work at hand deals with is that of the fullerene. While the hexagonal lattice in graphene is flat, the introduction of pentagons in the lattice creates curvature in the structure. Fullerenes are molecules that form closed polyhedron structures with  $n_h$  hexagons and exactly 12 pentagons, with some examples illustrated in figure 2.1 ranging from the smallest possible  $C_{20}-I_h$  with 20 atoms to a large nano-tube  $C_{1152}-D_{6d}$  with 1152 atoms. The fact that exactly 12 pentagons close a structure can be shown using Euler's formula<sup>1</sup>. Many of the mathematical properties for the fullerene structures has been mathematically described long before its discovery. This is in part within polyhedral combinatorics, that structures such as Goldberg polyhedra has been described since the 1930's, in which the icosahedral fullerenes are examples.

While in theory the possible number of fullerenes is infinite, there are limitations to which structures will create chemically stable molecules. One aspect of the geometries that generally yield stable structures is the isolated pentagon rule(IPR)[12]. This states that fullerenes where no pentagons are adjacent to each other are generally more stable. This is due to the distortion of the fullerene cage away from a flat structures being directly linked to the molecules thermodynamic instability, which the IPR limits. Figure 2.1 reveals that  $C_{20}-I_h$  obviously does not obey the IPR since it is solely constructed from pentagons, making the  $C_{60}-I_h$  the smallest fullerene to obey the rule.  $C_{60}-I_h$  and  $C_{20}-I_h$  are among the few fullerenes to have icosahedral symmetry. Meaning the structures have 60 rotational symmetries and a total symmetry order of 120 operations when accounting for reflectional operations.

### 2.1.1 Gaussian Curvature

Gaussian curvature is an important quantity when understanding the geometrical aspects of fullerenes. Gaussian curvature is defined as being the product of the the two principal curvatures at point  $p$ . For each point this will be the product of the maximal curvature  $k_1$  and minimal curvature  $k_2$ , when taking all direction on the surface into account i.e  $K = k_1 k_2$ . The Gaussian curvature is an intrinsic property

<sup>1</sup>which reads  $N_v - N_e + N_f = 2$  with  $N_v$ ,  $N_e$  and  $N_f$  being the number of vertices, edges and faces



**Figure 2.1:** Examples of fullerenes. (a) and (b) are the spherical  $C_{20}-I_h$  and  $C_{60}-I_h$ ,  $I_h$  referring to icosahedral symmetry, (c) is the cubic shaped  $C_{140}-D_{3h}$ , (d) is the menhir shaped  $C_{524}-C_1$ , (e) and (f) are the nano-tubes of  $C_{360}-D_{5h}$  and  $C_{1152}-D_{6d}$  respectively. From [6] with permission.

for all points on the surface, illustrated in figure 2.2 for shapes with different Gaussian curvatures. If one can wrap a 3-dimensional structure in a sheet without tearing or stretching, then the Gaussian curvature is zero everywhere on the surface, as is the case for the cylinder in figure 2.2. A positive Gaussian curvature will represent a closing surface such as a sphere. A negative value represents an opening surface such as a hyperboloid. Introducing a point with positive Gaussian curvature in a flat space, allows to be cut up and represented on a 2-dimensional surface, which is not the case for the negative case.

Through the Gaussian curvature  $K$  aspects of the topology of a surface can be understood using the Gauss-Bonnet theorem

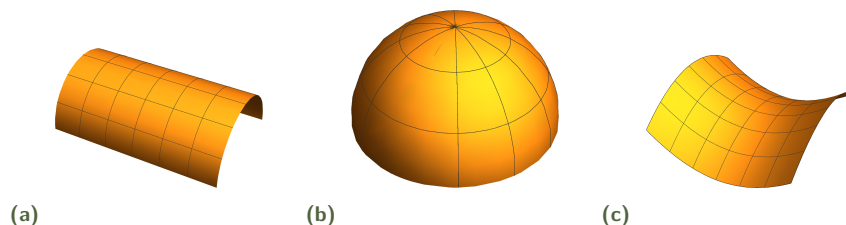
$$\int_S ds K(s) = 4\pi(1 - g), \quad (2.1.1)$$

where  $s$  is the oriented surface  $S$  and  $g$  as the genus of the surface. A sphere ( $g = 0$ ) therefore has total curvature of  $4\pi$ , while a torus ( $g = 1$ ) has 0. As shown in [6] each hexagons in fullerenes structures have Gauss curvature of 0 everywhere and are therefore flat. The pentagons however induce a positive curvature of  $2\pi/6$ . The discrete version of equation (2.1.1) reads

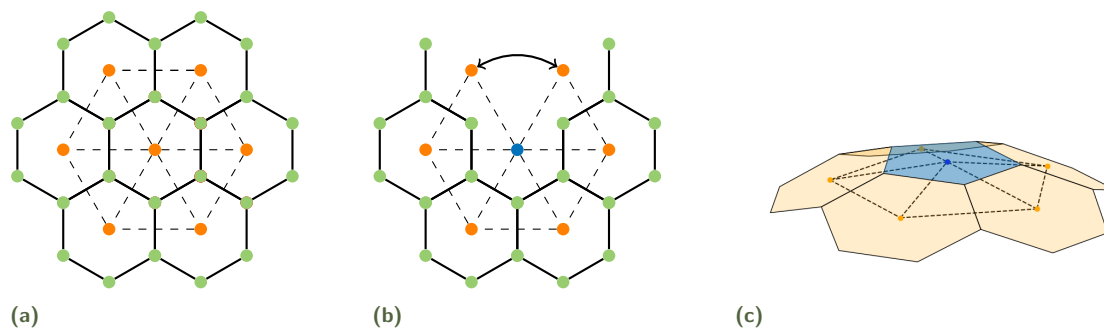
$$\sum_{i=1}^n K_i = 4\pi(1 - g). \quad (2.1.2)$$

and will therefore have 12 pentagons induce a total curvature of  $12 \cdot 2\pi/6 = 4\pi$ , in order to yield a closed structure. A visualization of the curvature produced by introducing a pentagon into a honeycomb

lattice is shown in figure 2.3. Here the triangular wedge removed in figure 2.3b has the angle  $2\pi/6$  yielding the curved structure in figure 2.3c. Figure 2.3 also depicts the dual representation, which in the context of fullerenes can be quite advantageous. The surfaces in the dual representation consist of equilateral triangles where each triangular face corresponds to an atom placed in the center. Note also that each vertex is placed at the center of a hexagon or pentagon. This means that a dual vertex representing a hexagon will outline 6 triangles, and 5 triangles for pentagons. It can also be seen from figure 2.3a that Voronoi cells for the dual reconstructs the polygonal shapes. This point is quite important and will be emphasized later on.



**Figure 2.2.:** Surfaces where all points have Gauss curvature of (a) zero, (b) positive and (c) negative. From [11] with permission.



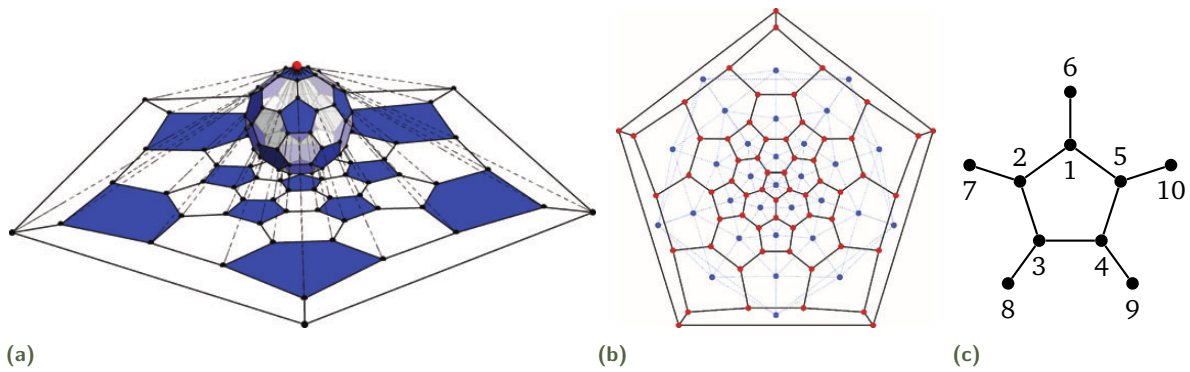
**Figure 2.3.:** (a) A hexagonal honeycomb lattice with vertices/atoms in green and its dual representation in orange consisting of connected triangles. (b) Restructuring the lattice to have a pentagon in the middle by effectively removing a wedge of  $2\pi/6$  from the dual and merging the dual vertices connected by the arrow. (c) Illustration of the curvature induced by a pentagon following the operation in (b).

## 2.1.2 Graph Theory

The geometrical structures and connecting features of the bonds can be characterized mathematically by graph theory. A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  consists of vertices  $\mathcal{V}$  which in our case are the atoms, pairwise connected by edges  $\mathcal{E} \subset \{(x, y) | (x, y) \in \mathcal{V}^2 \text{ and } x \neq y\}$  which corresponds to the neighbouring bonds. An important distinction is the difference between an undirected and a directed graph. An undirected graph have unordered pairs i.e. no difference between edges  $\mathcal{E} = (x, y)$  and  $\mathcal{E}' = (y, x)$ . In the case of a directed graph the edges have orientations. Our case is that of the 3-regular graph where all vertices are connected to three other vertices, also known as a cubic graph.

If it is possible to map a graph onto a 2-dimensional space, such that no intersections of edges occur, the graph is said to be planar. A drawing, where vertices are given 2-dimensional coordinates and connected by the non-intersecting same edges, is called a planar embedding of the graph. Creating embeddings for fullerenes can be done through various projections. The approach of the Schlegel projection can be performed, starting by placing a plane beneath the 3-dimensional structure. Then by adding a point  $p$  slightly above the molecule and drawing lines from  $p$  through each vertex onto the plane, shown in

figure 2.4a. To perform the Schlegel projection one needs the 3-dimensional structure, however this projection occasional yield crossing edges<sup>2</sup>.



**Figure 2.4.:** (a) Constructing the plane graph for the Buckminsterfullerene using the Schlegel projection where the red dot represents the added point  $p$ . (b) The produced plane graph where the red shows atoms, while blue is the dual representation found by calculating the mean for coordinate for each polygon. (c) a labelled pentagon used for the adjacency matrix which represents the graph. (a) and (b) from [6] with permission.

An easy approach to represent the connectivity in a graph is through an Adjacency matrix which will be denoted  $C$ .  $C$  is a simple symmetric matrix, where the entry is  $C_{ij} = 1$  if  $(i, j) \in \mathcal{E}$  i.e. if  $i$  and  $j$  are vertices connected by an edge, if  $(i, j) \notin \mathcal{E}$  then the entry is  $C_{ij} = 0$ . Using the labelled pentagon in figure 2.4c the adjacency matrix takes the form

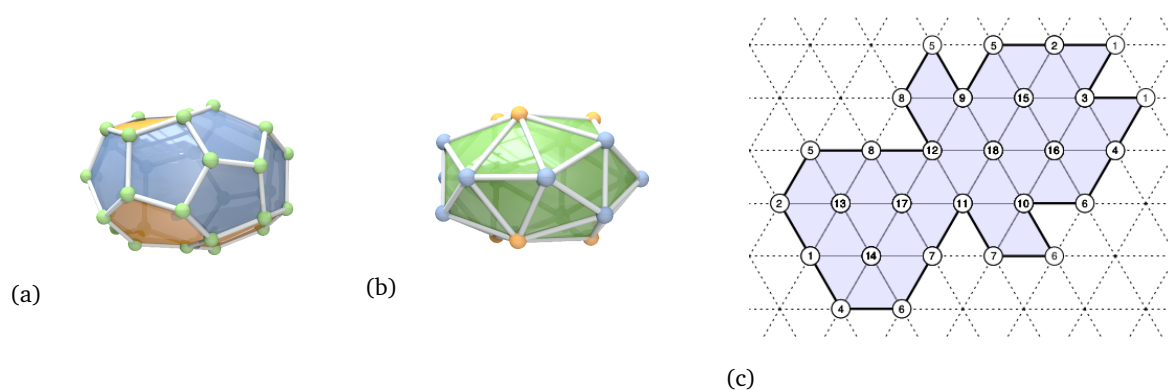
$$C = \begin{pmatrix} 0 & \textcircled{1} & 0 & 0 & \textcircled{1} & \textcircled{1} & 0 & 0 & 0 & 0 \\ \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 \\ 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & \textcircled{1} & 0 & 0 & 0 & \textcircled{1} & 0 \\ \textcircled{1} & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & \textcircled{1} \\ \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad C_s = \begin{pmatrix} 2 & 5 & 6 \\ 1 & 3 & 7 \\ 2 & 4 & 8 \\ 3 & 5 & 9 \\ 1 & 4 & 10 \\ 1 & .. & .. \\ 2 & .. & .. \\ 3 & .. & .. \\ 4 & .. & .. \\ 5 & .. & .. \end{pmatrix}. \quad (2.1.3)$$

Here the left matrix show the  $N \times N$  adjacency matrix containing all necessary graph information. Next to it is a sparse representation  $C_s$  of size  $N \times 3$ . The order in which the labels appear in sparse representation it not defined by the representation, as is the case for the dense representation. The orientation of the surfaces can therefore be decided by the order in which the vertices appear in  $C_s$ . Thus the sparse representation actually contain more information than its dense counterpart and will be important when dealing with finite element methods.

<sup>2</sup>A more stable projection scheme which does not require the initial 3-dimensional geometry is the Tutte-embedding. This will through the adjacency matrix always produce a plane graph.

### 2.1.3 2-Dimensional Unfolding of Dual Representation

The dual representation of fullerenes consist solely of equilateral triangles. This allows for an unfolding to a 2-dimensional plane consisting of equilateral triangles. This is done through Eisenstein integers which are complex numbers of the form  $z = a + b\omega$  with  $\omega = e^{i2\pi/6}$ . These integers make up a triangular grid consisting of equilateral triangles called the Eisenstein plane.  $(a, b)$  can be thought of as a coordinate pair defining a vertex, with each vertex being connected to 6 triangles. For each pentagon one of these triangles will not be included, and edges can be glued together as in figure 2.3. All pentagon nodes must therefore lie on the outer rim of the unfolding. All interior vertices in an unfolding are part of a hexagonal mesh in which line and angles behave Cartesian. An example of a non-unique folding for  $C_{32}-D_{3h}$  is illustrated in figure 2.5c. All the numbered labels appearing more than once are the same vertex in a 3-dimensional representation, and show how the unfolding can be folded into the fullerene. Note all vertices defining pentagons only define 5 triangles. These 2-dimensional unfoldings are a convenient way to visualize the solution to whatever differential equation that is investigated on the manifold surface.

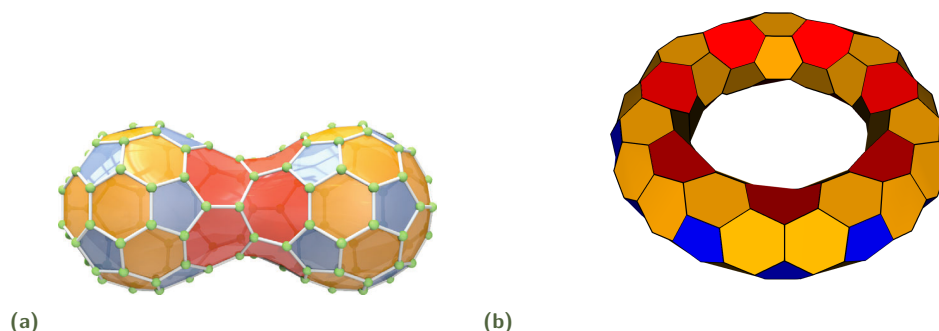


**Figure 2.5.:** (a) The  $C_{32}-D_{3h}$  fullerene its corresponding dual shown in (b). This allows for non unique 2-dimensional unfolding on the Eisenstein plane with an example given in (c), where each labelled intersection define a dual vertex. From [11] with permission.

### 2.1.4 Generalization to Fullerooids

Fullerooids are structures in which it is maintained that atoms only form three bonds and form closed surfaces with regular polygons as faces, but allow for other polygons than pentagons and hexagons. Since polygons such as heptagons and octagons induce a negative Gauss curvature which curves away from the tangent plane in two different directions it will thereby open the structure. The structures are therefore still within cubic graph theory which allow for many interesting and somewhat weird possible shapes, where once again chemical stability is not a guarantee by any means. A few examples are illustrated in figure 2.6 which using pentagons, hexagons and heptagons construct peanut shape fullerooid ( $g = 0$ ) and a torus ( $g = 1$ ). A heptagon induces a negative curvature  $-2\pi/6$  each heptagon and pentagon therefore cancel out the total Gauss curvature they add. A fullerooid of ( $g = 0$ ) consisting of pentagons, hexagons and heptagons as the peanut shape in figure 2.6a, must therefore have exactly 12 more pentagons than heptagons. For the torus in figure 2.6b with ( $g = 1$ ) through have an equal number of pentagons and heptagons to form the closed structure. The torus illustrates the concept of negative and positive curvature quite well, with all pentagons being on the outer rim "closing" the fullerooid, while the heptagons all lie in the inner ring. The shape is constructed using chemical stable

coordinates computed in [13], which is why not all polygons seem to lie in a perfect plane, a more thorough description will be given in section 3.6.



**Figure 2.6.:** (a) A  $C_{120}$  peanut shaped fulleroid with 10 heptagons and 22 pentagons. (b) A  $C_{168}$  torus shaped fulleroid consisting of 14 pentagons and 14 heptagons with chemically stable coordinates from, explained further in section. (a) From [6] with permission.

## 2.2 Finite Element Method

As one of the fundamental cornerstones in physics we have differential equations. As mentioned in the introduction the challenge of constructing a 2-dimensional DFT method on the manifold surfaces involve solving several PDEs. We wish to obtain these solutions solely from the sparse adjacency matrix without ever needing the 3-dimensional embedding of the faces. This will be done using the finite element method(FEM) also referred to as finite element analysis(FEA). FEM is a quite common approach in the engineering arts, e.g. used for computing stress-strain relations in complicated structures. What makes FEM especially suited for this work is, that solutions can be computed entirely from information of a fullerenes graph representation.

### 2.2.1 FEM Overview

Implementation of a FEM can at the very core be boiled down to a series of steps namely:

- Express the differential problem in a variational formulation in infinite-dimensional space  $V$ .
- Reformulate the variational formulation in finite-dimensional subspace  $\tilde{V}_h \subset V$ .
- Construct the function space  $\tilde{V}$  by meshing the domain and defining basis functions on each cell in the mesh.
- Assembly of the relevant basis matrices and vectors required by the variational formulation of the problem.
- Solution of the system of equations.

This sections will present the analysis in the above order for a specific differential equation. The reader should hopefully not find it to difficult to derive and construct the FEM for another given problem. While full theoretical frameworks of each of these elements are rich subjects each with a considerable amount of nuances and relevant material, this section will aim to address the most important concepts of the analysis. For a comprehensive study of the mathematical theory of the Finite Element Methods the reader is referred to Brenner & Scott[14]



## 2.2.2 Variational Formulation

Consider the problem where we let  $\Omega$  be a bounded domain in space  $\mathbb{R}^n$  with boundary  $\Gamma$ . Given a boundary value problem of

$$\begin{aligned} -\nabla^2 u &= f & \text{in } \Omega, \\ u &= g_0 & \text{on } \Gamma. \end{aligned} \quad (2.2.1)$$

The problem states that given an input function  $f \in C(\Omega)$  with a Dirichlet boundary condition of value  $g_0$  we must find  $u \in C^2(\Omega)$ . Here the notation used for spaces of smooth functions are

$$C(\Omega) := \{f : \Omega \rightarrow \mathbb{R}^n \mid f \text{ continuous on } \Omega\} \quad \text{and} \quad (2.2.2)$$

$$C^k(\Omega) := \{f \in C(\Omega) \mid D^\alpha f \in C(\Omega), \text{ for } |\alpha| \leq k\}, \quad (2.2.3)$$

with  $D^\alpha$  being a shorthand notation for an arbitrary differential operator

$$D^\alpha := \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \dots \partial x_n^{\alpha_n}}. \quad (2.2.4)$$

The stated problem is the Poisson equations in a  $n$ -dimensional space. This elliptic PDE describes many different physical phenomenon such as an electrostatic potential caused by a charge density and a gravitational potential arising from a mass with a defined density in Newtonian gravity. The point-wise evaluation needed for the system in the above problem have strict continuity requirements and can not account for discontinuous behaviour. In addition to this  $f$  might be continuous but not  $\alpha$ -differentiable, e.g.  $f = |x|$  for which the differential is undefined in  $x = 0$ . The problem can be expressed in a corresponding weak form, which does not require the point-wise evaluation of the derivative needed in the strong form. For the ordinary calculus derivative requires

$$\frac{d}{dx} f(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (2.2.5)$$

to be point-wise defined for all  $x$ . The weak derivatives can instead be defined in terms of inner products on spaces of  $L_{loc}^1$ <sup>3</sup>, i.e. the function is only assumed to be locally integrable.

**Definition 1** (Weak derivative). Given a function  $f \in L_{loc}^1(\Omega)$  with a weak derivative  $D_w^\alpha f$  if there exists a function  $g \in L_{loc}^1(\Omega)$  such that

$$\int_{\Omega} g(x)\phi(x)dx = (-1)^{|\alpha|} \int_{\Omega} f(x)D^\alpha \phi(x)dx \quad \forall \phi \in C_0^\infty. \quad (2.2.6)$$

If such  $g$  exists, we define  $D_w^\alpha f = g$ .

Which generally allows for defining derivatives for a much larger class of functions compared to the common derivative. This is the case since the integral formulation allow us to solve the differential equations using Hilbert spaces.

Continuing from the stated Poisson problem, the weak form is expressed in integrals with respect to some test function  $v$ . Let the test function be any sufficiently regular function with the boundary

<sup>3</sup> $L_{loc}^1(\Omega)$  being the set of locally integrable functions, where the Lebesgue integral is finite on all compact subsets of  $\Omega$ .

behaviour of  $v(\Gamma) = 0$  then applying the test function to each side of equation (2.2.1) and integrating over the domain i.e.

$$-\int_{\Omega} v \nabla^2 u \, dx = \int_{\Omega} f v \, dx. \quad (2.2.7)$$

or equivalently in bra-ket notation

$$-\langle v | \nabla^2 | u \rangle = \langle v | f \rangle. \quad (2.2.8)$$

As it will now be shown the problem can be stated to depend on  $\nabla u$  instead of  $\nabla^2 u$  allowing for weaker regularity requirements on  $u$ . Rewriting the first term in equation (2.2.7) using  $\nabla \cdot (v \nabla u) = v \nabla^2 u + \nabla v \nabla u$  leaves

$$-\int_{\Omega} v \nabla^2 u \, dx = \int_{\Omega} \nabla v \cdot \nabla u \, dx - \int_{\Omega} \nabla \cdot (v \nabla u) \, dx. \quad (2.2.9)$$

The second term in equation (2.2.9) can then be evaluated from the integral over the domain  $\Omega$  to an integral over the boundary  $\Gamma$  using Green's theorem which for a vector field  $g$  is  $\int_{\Omega} \nabla \cdot g \, dx = \int_{\Gamma} g \cdot \mathbf{n} \, ds$  where  $\mathbf{n}$  is the surface normal. Following from equation (2.2.9) we now have

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\Omega} \nabla \cdot (v \nabla u) \, dx = \int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\Gamma} (v \nabla u) \cdot \mathbf{n} \, dx. \quad (2.2.10)$$

For the problem at hand the last boundary integral above vanishes due to the essential boundary condition on  $v$  which stated  $v(\Gamma) = 0$  thus leaving

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx = \int_{\Omega} f v \, dx \Leftrightarrow A(u, v) = F(v), \quad (2.2.11)$$

where the last shorthand notation  $A(u, v)$  and  $F(v)$  for the integrals are introduced. Note that the weak form in comparison to the strong form, which required  $u \in C^2(\Omega)$ , now has transformed one of the differential operators onto the test function  $v$ . Defining the function space  $V$  for which  $v \in V$

$$V = \{v : A(v, v) < \infty, F(v) < \infty \text{ and } v(\Gamma) = 0\} \quad (2.2.12)$$

then the solution to (2.2.1) in the weak formulation is specified by

$$\text{find } u \in V \text{ such that } A(u, v) = F(v) \quad \forall v \in V \quad (2.2.13)$$

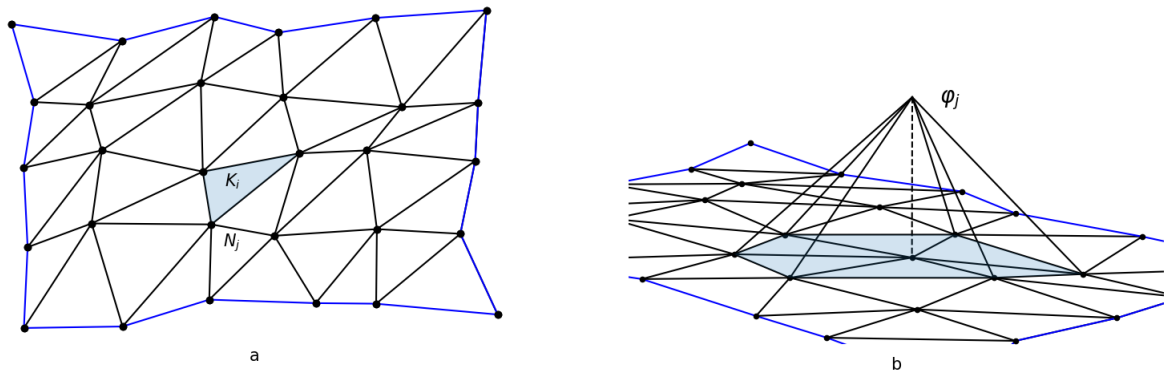
which is the problem fully reformulated to a weak form.

## 2.2.3 Ritz-Galerkin Approximation

Now that an infinite-dimensional large space  $V$  is defined we wish to formulate the same problem but in a finite-dimensional subspace of  $V$  i.e we want to construct  $\tilde{V} \subset V$ . With the subspace  $\tilde{V}$  a discrete approximation to (2.2.13) reads

$$\text{find } \tilde{u} \in \tilde{V} \text{ such that } A(\tilde{u}, \tilde{v}) = F(\tilde{v}) \quad \forall \tilde{v} \in \tilde{V}. \quad (2.2.14)$$

The accuracy of the approximate solution therefore depends of the size of the subspace  $\tilde{V}$ . The discretization allows for stating the problem as a system of linear equations thus computable. The FEM



**Figure 2.7.:** (a) Triangular mesh of a 2-dimensional polygonal domain bounded by the blue lines, and with  $K_i$  and  $N_j$  respectively denoting the shaded triangle and a vertex. (b) A linear basis function  $\varphi_j$  for vertex  $N_j$  on the constructed mesh where the shaded area highlight the triangles that share  $N_j$ .

uses the Ritz-Galerkin approximation by creating a mesh on the geometrical domain  $\Omega$  and assigning piece-wise polynomial functions to the mesh.

## 2.2.4 Discretization of Domains

We wish to construct a mesh of  $\Omega$  i.e. discretize the domain  $\Omega$  and equip it with a basis-set of piece-wise polynomial functions. A mesh of a given domain  $\Omega$  signify that the domain is decomposed into a collection of  $n$  smaller non-overlapping cells. These will usually have a simple geometrical structure. Thus denoting the individual cells  $T = K_1, \dots, K_n$  yields a domain of  $\Omega = K_1 \cup K_2 \dots \cup K_n = \bigcup_{K \in T_h} K$ . Cells are convex polytopes<sup>4</sup> with the dimensionality depending on the problem. The cells in a 1-dimensional problem are intervals, in 2-dimensions we have convex polygons such triangles, square etc. and for a 3-dimensional problem the structures are convex polyhedra such as tetrahedra, cubes etc. Cells are connected to neighbouring cells at the vertices. Thus, if two 2-dimensional cells share an intersect in some way it must be either a full shared edge or a shared vertex; in a 3-dimensional setting they share a vertex, edge or face. A 2-dimensional triangular mesh (triangulation) created for a domain can be seen in figure 2.7, note no vertices lie at the edge of another triangle.

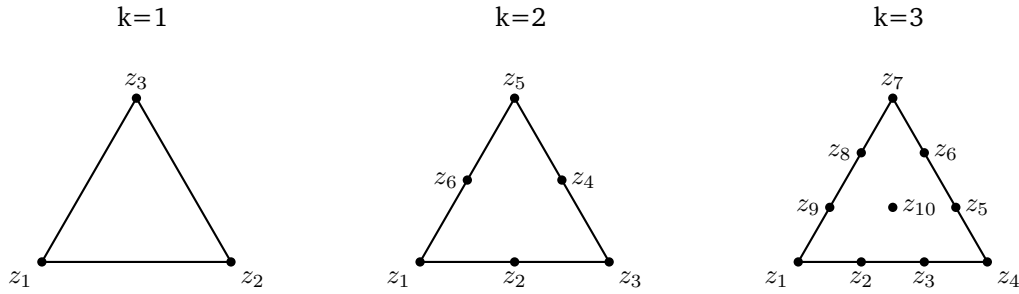
The next step is to equip each cell with polynomial functions. Taking the instance of equipping the triangulation seen in figure 2.7 with simple linear polynomial functions gives each triangle 3 shape functions each defined by a node  $z_i$ . In the case of linear basis functions each vertex  $N_i$  will coincide with node  $z_i$ . Each node  $z_i$  is associated with a basis function  $\varphi_i$  with the function value of 1 at the the corresponding node and 0 at all the other nodes i.e. a Kronecker Delta function

$$\varphi_i(z_j) = \delta_{ij} = \begin{cases} 1, & j = i \\ 0, & j \neq i. \end{cases} \quad (2.2.15)$$

This is illustrated in figure 2.7 which shows the linear functions vanishing on all triangles that does not share the vertex  $N_j$ . In the linear case the nodes and vertices coincidence and can be used interchangeably. This is however not the case for higher polynomial degrees. For triangles the number of linearly independent shape functions on a triangle is  $(k+1)(k+2)/2$  with  $k$  being the polynomial degree. Equipping a triangle cell with quadratic polynomial degree therefore has six nodes  $\{x_1, x_2, \dots, x_6\}$  which

<sup>4</sup>The general n-dimensional version of polygons and polyhedra.

obey equation (2.2.15). Three of the  $N_j$  vertices therefore define the corners making up a triangle  $K_i$ , while the nodes  $z_k$  define the basis functions on triangle  $K_i$ . Figure 2.8 has illustration of the placement of the nodes for linear, quadratic and cubic polynomial degree for an equilateral triangle. The number of individual nodes on a cell are called the local degrees of freedom and will be denoted  $n_{dof}$ , while the number of nodes present in the entire mesh is called the global degrees of freedom and denoted  $N_{dof}$ .  $n_{dof}$  therefore only depends on the polynomial degree, while  $N_{dof}$  not only depend on the number of  $n_{dof}$  but also the total number of cells in the mesh.



**Figure 2.8.:** The necessary nodes for a linear, quadratic and cubic polynomial implementation with an equilateral triangle as the cell structure. The number of points is  $(k + 1)(k + 2)/2$  with  $k$  polynomial degree, thus yielding  $n_{dof} = \{3, 6, 10\}$ . Note node  $z_{10}$  is uniquely confined to the triangle while a mesh with adjacent triangles has all other nodes included in several triangles.

Having created a mesh of  $\Omega$  and defined the shape functions accordingly, we can finally reformulate the problem to a computable linear system of equations. We start by expanding  $\tilde{\mathbf{u}}$  and  $\tilde{\mathbf{v}}$  in terms of the basis onto all  $N_{dof}$  nodes

$$\tilde{\mathbf{u}} = \sum_{i=1}^{N_{dof}} \tilde{u}(z_i) \varphi_i \quad \text{and} \quad \tilde{\mathbf{v}} = \sum_{j=1}^{N_{dof}} \tilde{v}(z_j) \varphi_j. \quad (2.2.16)$$

Introducing the shorthand notation of  $\tilde{u}(z_i) = \tilde{u}_i$  and  $\tilde{v}(z_j) = \tilde{v}_j$  for the sake of readability. Using the basis function expansion for  $\tilde{\mathbf{v}}$  in the shorthand integral notation of the problem (2.2.14) yields

$$A \left( \tilde{\mathbf{u}}, \sum_{j=1}^{N_{dof}} \tilde{v}_j \varphi_j \right) = F \left( \sum_{j=1}^{N_{dof}} \tilde{v}_j \varphi_j \right) \quad (2.2.17)$$

$$\Leftrightarrow \sum_{j=1}^{N_{dof}} \tilde{v}_j A(\tilde{\mathbf{u}}, \varphi_j) = \sum_{i=j}^{N_{dof}} \tilde{v}_j F(\varphi_j) \quad (2.2.18)$$

$$\Rightarrow A(\tilde{\mathbf{u}}, \varphi_j) = F(\varphi_j) \quad \text{for } j = 1, \dots, N_{dof}. \quad (2.2.19)$$

Now the same procedure by introducing the expansion of  $\tilde{\mathbf{u}}$  to the l.h.s of eq. (2.2.19)

$$A \left( \sum_{i=1}^{N_{dof}} \tilde{u}_i \varphi_i, \varphi_j \right) = F(\varphi_j) \quad (2.2.20)$$

$$\Rightarrow \sum_{i=1}^{N_{dof}} \tilde{u}_i A(\varphi_i, \varphi_j) = F(\varphi_j). \quad (2.2.21)$$

Which can be written as a problem of matrices and vectors using

$$A_{ij} = A(\varphi_i, \varphi_j) = \int_{\Omega} \nabla \varphi_i \cdot \nabla \varphi_j \, dx \quad (2.2.22)$$

and

$$f_i = F(\varphi_i) = \int_{\Omega} f \varphi_i \, dx \quad (2.2.23)$$

thus leaving a problem of

$$\mathbf{A} \tilde{\mathbf{u}} = \mathbf{f}. \quad (2.2.24)$$

Where the matrix  $\mathbf{A}$  is of size  $N_{dof} \times N_{dof}$  but sparse, and the vector of length  $N_{dof} \times 1$ . The above linear system can then be solved for the vector  $\tilde{\mathbf{u}}$  where  $\tilde{\mathbf{u}} = (\tilde{u}(z_1), \dots, \tilde{u}(z_{N_{dof}}))$ . The matrix  $\mathbf{A}$  is often referred to as the stiffness matrix in FEM and will henceforth be denoted  $\mathbf{W}$ . Note that the overlap integral of the stiffness matrix in equation (2.2.22) yields a sparse system due to the finite support of the basis functions; i.e if nodes  $\varphi_i$  and  $\varphi_j$  do not share a triangle, the integral is 0. One of the strengths of the FEM is control over the the function space which can be expanded quite easily. This can be achieved by either a finer mesh on the domain or higher polynomial degree on the cells.

The open questions are now how one can effectively assemble the matrices and vectors in question and how to evaluate the integrals. These will both be addressed in section 3 along with the FEM software implementation and the necessary PDEs such that the details important to the specific problems are not lost.

## 2.3 The Quantum Mechanical Many-Body Problem

In a quantum mechanical description the behaviour of the wave function  $|\Psi(t)\rangle$  govern all physical properties whether it represents a single particle or an ensemble. The wave function in the non-relativistic limit obey the Schrödinger equations which in the time-dependent case reads

$$H|\Psi(t)\rangle = i\hbar\frac{\partial}{\partial t}|\Psi(t)\rangle, \quad (2.3.1)$$

where  $i$ ,  $\hbar$  and  $H$  are the imaginary unit for complex numbers, the reduced Planck constant and the Hamiltonian operator associated with the quantum system respectively. If the Hamiltonian of the problem at hand is time-independent, equation (2.3.1) takes a simpler form, namely the time-independent Schrödinger equation:

$$(T + V)|\Psi\rangle = E|\Psi\rangle, \quad (2.3.2)$$

where the kinetic and potential operators  $T$  and  $V$  acts on the quantum state yielding the energy  $E$  of the particle/ensemble, i.e. the wave function is an eigenfunction to the Hamilton operator  $H = T + V$  with the energy  $E$  as the eigenvalue. Throughout this work many-body and single-body operators will be denoted by upper-case and lower-case respectively. For a single particle system in a potential  $v$ ,  $h$  takes the familiar form  $h = \left[ \frac{-\hbar^2}{2m}\nabla^2 + v \right]$ , where  $m$  is the mass of the particle and  $\nabla^2$  is the Laplace operator. Solving this analytically for different potentials is possible and yield some spectacular solutions. Although there are limitations to the number of analytical solutions we can generally produce solutions to 1-particle problems numerically exact to a desired precision.

Going from the single particle to the concept of the many-body problem. Once again the Schrödinger equation with an appropriate many-body Hamiltonian govern all behaviour, such as a collection of atoms forming matter. In theory this would encompass all types of ensembles and behaviour in all phases whether it be a crystal, ferrormagnet, superconductor etc. Taking equation (2.3.2) with an appropriate, somewhat daunting Hamiltonian for a collection of  $N_e$  electrons and  $N_n$  nuclei reads

$$H\Psi = [T_e + T_n + U_{ee} + U_{nn} + U_{en} + V_{ext}]\Psi = E\Psi, \quad (2.3.3)$$

where  $T_e$  and  $T_n$  are the kinetic energy operators for the electrons and nuclei respectively which are written as a sum over  $N_e$  and  $N_n$  i.e.

$$T_e = -\sum_{i=1}^{N_e} \frac{\hbar^2}{2m_e} \nabla_i^2 \quad \text{and} \quad T_n = -\sum_{I=1}^{N_n} \frac{\hbar^2}{2m_I} \nabla_I^2. \quad (2.3.4)$$

In the above, the Laplace operator  $\nabla_i^2$  and  $\nabla_I^2$  acts on a the  $i^{th}$  and  $I^{th}$  electron with mass of  $m_e$  and  $m_I$ . Note the subscript in  $m_I$  as the nuclei can have varying masses. The operators accounting for the Coloumb repulsion and attraction which is exerted between all particles present is accounted for as well. The two-body nature of each interaction is divided into  $U_{ee}$  as the electron-electron repulsion,  $U_{nn}$  as

the nuclear-nuclear repulsion and  $U_{en}$  as the electron-nuclear attraction. The sum of these potential terms define the internal potential and take the form

$$W \equiv U_{ee} + U_{nn} + U_{en} = e^2 \sum_{i=1}^{N_e} \sum_{j=i+1}^{N_e} \frac{1}{|\mathbf{x}_i - \mathbf{x}_j|} + e^2 \sum_{I=1}^{N_n} \sum_{J=I+1}^{N_n} \frac{Z_I Z_J}{|\mathbf{X}_I - \mathbf{X}_J|} \quad (2.3.5)$$

$$-e^2 \sum_{I=1}^{N_n} \sum_{i=1}^{N_e} \frac{Z_I}{|\mathbf{X}_I - \mathbf{x}_i|}, \quad (2.3.6)$$

with the elementary charge  $e$ ,  $Z_I$  as the atomic charge of nucleus  $I$ ,  $\mathbf{x}_i$  and  $\mathbf{X}_I$  as the electronic and nuclear coordinates respectively. Lastly  $V_{ext}$  is the external potential in which the system is situated in, which acts separately on each particle, for instance a molecule in an electric field.

The system to solve is then a partial differential equation with  $3(N_e + N_n)$  degrees of freedom that are non-separable due to the two-body nature of the Coulomb interaction<sup>5</sup>. This problem is more or less impossible to solve analytically, only a few simple systems can be solved like H and the ions  $\text{He}^+$ ,  $\text{Li}^{2+}$ ,  $\text{Be}^{3+}$ ,  $\text{B}^{4+}$  and the molecule  $\text{H}_2^+$ . Common for these are a single core with a single electron, which still require require the use of the Born-Oppenheimer approximation to achieve an analytical solution.

### 2.3.1 Born-Oppenheimer Approximation

Focusing on molecules and for the following ignore any external fields so  $V_{ext} = 0$ . The non-separability of equation (2.3.3) means that the wave function can not be split into a nuclear and electronic part i.e.  $\Psi(\mathbf{x}, \mathbf{X}) = \Phi(\mathbf{x})\Theta(\mathbf{X})$  where  $\Phi(\mathbf{x})$  and  $\Theta(\mathbf{X})$  respectively refer to the electronic and nuclear part of the wave function is not possible. However the time scale at which the kinetic energy for the electrons and nuclei operate differ greatly due to difference in mass. The mass ratio between a proton and an electron is 1:1836, thus classically speaking an equal force applied to the two will cause a significantly greater acceleration on the electron considering Newton's second law. In a quantum mechanical formalism the electrons wave functions adjust instantaneously with whatever slow nuclei dynamical behaviour the molecule exhibits. In the Born-Oppenheimer approximation the electronic wave function is solved with the nuclei assumed fixed in space. Assuming nuclei positions  $\{\mathbf{X}'_1, \dots, \mathbf{X}'_{N_n}\}$  a priori and thereby ignoring the kinetic nuclei kinetic operator  $T_n$ , the Schrödinger equation for the electronic wave equations reads

$$[T_e + U_{ee} + U_{nn} + U_{en}]\Phi^{(k)}(\mathbf{x}, \mathbf{x}') = E_e^{(k)}(\mathbf{X}')\Phi^{(k)}(\mathbf{x}, \mathbf{X}'), \quad (2.3.7)$$

in which the notation for the electronic wave function  $\Phi^{(k)}(\mathbf{x}, \mathbf{X}')$  denote  $k'$ th eigenstate being parametrically dependent on  $\mathbf{X}'$  with corresponding energy  $E_e^{(k)}(\mathbf{X}')$ . Note  $U_{nn}$  depends on the fixed coordinates  $\mathbf{X}'$  and is just a constant term based on the configuration. By varying the configuration of  $\mathbf{X}'$  the nuclear Schrödinger equation can be solved with  $E_e^{(k)}(\mathbf{x})$  acting as an potential surface in which the nuclei can move. While electrons tend to have quite wavefunctions of a delocalized nature the nucleus wave functions are localized partly due to the large mass. Recalling that the thermal wavelength at temperature  $T$  is  $\lambda_T = (\hbar^2/2Mk_bT)^{1/2}$ , thus showing proportionality to the mass of  $\lambda_T \propto M^{-1/2}$ . The probability cloud for nucleus wave-function will therefore only have small variations around the classical position. By this reasoning the nuclei behaviour can be approximated by treating them as classical particles. The Born-Oppenheimer approximation is a vital step in computational chemistry

<sup>5</sup>On top of this the electronic part of the wave function must be anti-symmetric due to the  $\frac{1}{2}$ -spin fermion nature of electrons. While the nuclei can either have integer nuclear spin or half-integer nuclear spin which are characterized by boson and fermion behaviour respectively, demanding symmetric or anti-symmetric with respect to exchange of nuclear variables.

in minimizing computation. Now the problem has been made simpler by using stationary nuclei for solving the electronic wave equation, we however still have  $3N_e$  degrees of freedom with correlation between all electrons.

## 2.3.2 Density Functional Theory

Restating the problem and viewing the electronic density as the fundamental property of a system is at the heart of density functional theory, henceforth referred to as DFT. DFT is an extremely popular computational method in quantum chemistry for determining properties of many body systems. The motivation for this density based formulation is validated by the Hohenberg and Kohn theorems, which prove that the ground state energy  $E_0$  for an electronic system is completely determined by the electronic density  $\rho$ . Formulating the problem in a density manner instead of the full wave function reduces the complexity of the problem significantly. The  $N$  electron wave function depend on  $3N$  variables<sup>6</sup>, and the dimensionality of the system  $M$  increases combinatorially as  $\binom{M}{N}$  with the number of electrons  $N$ . We see that the density

$$\rho(\mathbf{x}) := N \int d^3x_2 \cdots \int d^3x_N |\Psi(\mathbf{x}, \mathbf{x}_2, \dots, \mathbf{x}_N)|^2, \quad (2.3.8)$$

however only depend on 3 spatial variables and therefore independent of the number of electrons.

A reader, unfamiliar with the proceedings should perhaps have build some skepticism towards using  $\rho$  as a fundamental property instead of  $\Psi$ , since it would seem that we are losing information present in  $\Psi$ . For the purpose of intuition keep in mind that the normalization of the integral over the density yields the number of electrons i.e.  $N = \int \rho(\mathbf{x})d\mathbf{x}$  and that the localization and height of cusps in the density correspond to nuclear positions and nuclear charges.

For good measure a short definition of functionals might be in order. While a function maps a number  $x_1$  onto another number  $x_1 \rightarrow f(x_1)$ , a functional maps a function onto a real or complex number  $f(x) \rightarrow F[f]$ . Functionals will be denoted by square bracket e.g.  $F[f]$  is a functional of function  $f$ <sup>7</sup>.

In the proofs for the Hohenberg-Kohn theorems the Hamiltonian for the electrons reads  $H = T + V_{ext} + U_{ee}$ , here  $V_{ext}$  refer to being a potential external to the electrons and could include attraction from nuclei as well as fields external to the molecule. Potential operators will also be represented as a sum of the one-body operators i.e  $V_{ext} = \sum_{i=1}^N v_{ext}(\mathbf{x}_i)$  where the  $v_{ext}(\mathbf{x}_i)$  acts on the  $i$ 'th particle. The expectation  $V_{ext}$  is therefore  $V_{ext} = \langle \Phi' | V_{ext} | \Phi' \rangle = \langle \Phi' | \sum_{i=1}^N v_{ext}(\mathbf{x}_i) | \Phi' \rangle = \int \rho(\mathbf{x})v_{ext}(\mathbf{x})d\mathbf{x}$ .

## Hohenberg-Kohn Theorems

Since DFT lie at the heart of the work at hand a walk through of the mathematical foundation for the formulation is in order

**Theorem 1 (First Hohenberg-Kohn Theorem).** *The electronic density in a bound system is unambiguously determined by the external potential, apart from an additive constant.*

*Proof:* Assuming the opposite to hold, the electronic density does not unambiguously determine the external potential. Then it should be possible to find two potentials  $V_{ext}$  and  $V'_{ext}$  with an identical ground state density  $\rho$ . Let  $\Phi_0$  be the wave function of the ground state with the energy  $E_0$  and a Hamiltonian of  $H = T + V_{ext} + U_{ee}$ . Introducing a similar system denoted  $\Phi'$ ,  $E'_0$  and  $H'$ . We assume  $\Phi$

<sup>6</sup>If spin is included in the formulation  $\Psi$  depends on  $4N_e$  independent variables due to each electrons spin. This is taken into account in the density by summing over all spins in equation (2.3.8)

<sup>7</sup>To avoid confusing notation the function variable is omitted when denoting functionals, so if  $f$  is stated as a function of  $x$  the functional is denoted  $F[f]$  and not  $F[f(x)]$



and  $\Phi'$  give rise to an identical density  $\rho$  and that the Hamiltonians only differ in the external potential i.e.  $H' = T + V'_{ext} + U_{ee}$ . Then by virtue of the variational principle it follows

$$E_0 < \langle \Phi' | H | \Phi' \rangle = \langle \Phi' | H + H' - H' | \Phi' \rangle = \langle \Phi' | H' | \Phi' \rangle + \langle \Phi' | H - H' | \Phi' \rangle \quad (2.3.9)$$

$$= E'_0 + \langle \Phi' | V_{ext} - V'_{ext} | \Phi' \rangle = E'_0 + \int \rho(\mathbf{x}) [v_{ext}(\mathbf{x}) - v'_{ext}(\mathbf{x})] d\mathbf{x}, \quad (2.3.10)$$

where the last argument follows from the fact that the contribution from the external potential, can be evaluated explicitly in terms of the density by

$$V_{ext}[\rho] = \langle \Phi | V_{ext} | \Phi \rangle = \int \rho(\mathbf{x}) v_{ext}(\mathbf{x}) d\mathbf{x}. \quad (2.3.11)$$

Following with an identical reasoning for  $E'_0$  yields

$$E'_0 < E_0 - \int \rho(\mathbf{x}) [v_{ext}(\mathbf{x}) - v'_{ext}(\mathbf{x})] d\mathbf{x}. \quad (2.3.12)$$

Adding the inequalities from equations (2.3.9) and (2.3.12) yields a clear contradiction that reads

$$E_0 + E'_0 < E'_0 + E_0. \quad (2.3.13)$$

The above contradiction leads to the assumption that there can not exist two different external potentials that yield the same electronic density hence proving the theorem. This implies that  $\rho_0$  uniquely determines the full Hamiltonian of the system thus determining all properties derivable from solving the full N-body Schrödinger equation. The proof shows the existence of a function  $\rho_0 \rightarrow H$ , but leaves no explanation of how to compute it.

**Theorem 2** (Second Hohenberg-Kohn Theorem). *The ground state energy  $E_0$  can be derived from the density  $\rho$  using the variational principle. The electron density that provides the ground state energy is therefore the exact and unique<sup>8</sup> ground state density.*

Defining functional  $F[\rho']$  as

$$F[\rho'] \equiv \langle \Phi[\rho'] | T + U_{ee} | \Phi[\rho'] \rangle, \quad (2.3.14)$$

where the minimum is taken over all  $\Phi$  which produces  $\rho'$ . Then the second Hohenberg-Kohn theorem states the energy functional is

$$E[\rho'] = F[\rho'] + \int \rho'(\mathbf{x}) v_{ext}(\mathbf{x}) d\mathbf{x} \geq E_0 \quad (2.3.15)$$

and in the special case of  $\rho' = \rho_0$  then

$$E_0 = F[\rho_0] + \int \rho_0(\mathbf{x}) v_{ext}(\mathbf{x}) d\mathbf{x}. \quad (2.3.16)$$

Thus showing that the total energy can be expressed in full by the density. While theorem 1 and 2 formulate the mathematical basis for DFT, they however do not provide any help with approximating  $F[\rho]$ , and the exact minimization calculation in theorem 2 is quite impractical as well. The difficulty in

<sup>8</sup>Assumed to be a non degenerate problem

$F[\rho]$  is due to the electrons complicated interacting nature. To aid in approximating this we call upon the Kohn-Sham equations.

## Kohn Sham Theory

A general approach to approximating  $F[\rho]$  offered by Kohn and Sham, was to cast parts of the problem into a non-interacting electronic system. Applying that a system of non-interacting electrons is fully described by the anti-symmetric wave function

$$\Phi(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \phi_1(1) & \phi_2(1) & \cdots & \phi_N(1) \\ \phi_1(2) & \phi_2(2) & \cdots & \phi_N(2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(N) & \phi_2(N) & \cdots & \phi_N(N) \end{vmatrix}, \quad (2.3.17)$$

called the Slater determinant. Here  $\psi_i(j)$  refers to the  $i$ 'th one electron orbital with the  $j$ 'th spatial and spin components. With the Slater determinant the kinetic energy contribution can be calculated exactly by  $T = \langle \Phi(\mathbf{x}) | T | \Phi(\mathbf{x}) \rangle = -\frac{\hbar^2}{2m} \sum_{i=1}^N \langle \phi_i(\mathbf{x}) | \nabla^2 | \phi_i(\mathbf{x}) \rangle$ . The non-interacting  $T$  is obviously not equal to the kinetic contribution in the interacting case. But the difference between the two is however of a much smaller order than  $T$ . Introducing a fictitious effective potential in which the non-interacting particles move  $v_{KS}(\mathbf{x})$  namely the Kohn-Sham potential. In a system with  $N$  occupied orbitals the ground state density comes down to solving the equations

$$\hbar \phi_i = \left[ -\frac{\hbar^2}{2m} \nabla^2 + v_{KS}(\mathbf{x}) \right] \phi_i = \varepsilon_i \phi_i \quad \text{with} \quad \rho(\mathbf{x}) = \sum_{i=1}^N |\phi(\mathbf{x})_i|^2. \quad (2.3.18)$$

If an exact Kohn-Sham potential actually exist is unknown, but if it does it will be unique due to the first Hohenberg-Kohn theorem. The Kohn Sham ansatz therefore assumes that the Hamiltonian for a non-interacting system with an appropriate Kohn-Sham potential exists and has the same ground state density as the interaction system. The energy functional for the non-interacting system and for good measure the interacting system reads

$$E^{KS}[\rho] = T^{KS}[\rho] + \int \rho(\mathbf{x}) v_{KS}(\mathbf{x}) d\mathbf{x} \quad \text{and} \quad E[\rho] = F[\rho] + \int \rho(\mathbf{x}) v_{ext}(\mathbf{x}) d\mathbf{x} \quad (2.3.19)$$

Assuming electrons move in a mean field due to all electrons including itself, will have an energy functional  $U_{ee}^{mf}[\rho]$  that is the classical electrostatic energy for charge distributions

$$U_{ee}^{mf}[\rho] = \langle \Phi[\rho'] | U_{ee} | \Phi[\rho'] \rangle = \frac{1}{2} \int \int \frac{\rho(\mathbf{x}) \rho(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x} d\mathbf{x}'. \quad (2.3.20)$$

This mean-field approach does obviously not hold physically, but we now introduce a correction to the energy called the exchange-correlation functional to account for this. This functional is defined by subtracting the non-interacting kinetic energy and the mean field approximation from the  $F[\rho]$  in equation (2.3.14) i.e

$$E^{XC}[\rho] \equiv F[\rho] - U_{ee}^{mf}[\rho] - T^{KS}[\rho] \quad (2.3.21)$$

The aim of exchange-correlation functional is to correct for the two false assumptions made. One, that the electrons move independently, which is false due to not only repulsion but the Pauli exclusion principle as well. Two, that each electron repels itself through the electronic density.

With  $E_{XC}[\rho]$  defined the pressing matter is how the Kohn-Sham potential is obtained. Starting by setting the two ground state energies  $E^{KS}[\rho] = E[\rho]$  from equation (2.3.19) equal and isolating the Kohn-Sham potential one obtains

$$\int \rho(\mathbf{x})v_{KS}(\mathbf{x})d\mathbf{x} = \int \rho(\mathbf{x})v(\mathbf{x})d\mathbf{x} + F[\rho] - T^{KS}[\rho], \quad (2.3.22)$$

which with equation (2.3.21) becomes

$$\int \rho(\mathbf{x})v_{KS}(\mathbf{x})d\mathbf{x} = \int \rho(\mathbf{x})v(\mathbf{x})d\mathbf{x} + U_{ee}^{mf}[\rho] + E^{XC}[\rho]. \quad (2.3.23)$$

For us to evaluate the Kohn-Sham potential an approach to functional derivatives is needed. A general approach to functional derivatives can be found in appendix A of [15]. With a functional type of

$$F[\rho] = \int f(x, \rho, \rho^{(1)}, \rho^{(2)}, \dots, \rho^{(n)}) d\mathbf{x}, \quad (2.3.24)$$

where notation  $\rho^{(n)}(\mathbf{x}) = \frac{\partial^n \rho(\mathbf{x})}{\partial \mathbf{x}^n}$  the general functional derivative reads

$$\frac{\delta F}{\delta \rho} = \frac{\partial f}{\partial \rho} - \frac{d}{d\mathbf{x}} \left( \frac{\partial f}{\partial \rho^{(1)}} \right) + \frac{d^2}{d\mathbf{x}^2} \left( \frac{\partial f}{\partial \rho^{(2)}} \right) - \dots + (-1)^n \frac{d^n}{d\mathbf{x}^n} \left( \frac{\partial f}{\partial \rho^{(n)}} \right). \quad (2.3.25)$$

Isolating the Kohn-Sham potential can then be done by taking the functional derivative with respect to  $\rho(\mathbf{x})$  on both sides of the equation (2.3.23).

$$v_{KS}(\mathbf{x}) = \frac{\delta}{\delta \rho} \int \rho(\mathbf{x})v_{KS}(\mathbf{x})d\mathbf{x} \quad (2.3.26)$$

$$= \frac{\delta}{\delta \rho} \left( \int \rho(\mathbf{x})v(\mathbf{x})d\mathbf{x} + \frac{1}{2} \int \int \frac{\rho(\mathbf{x})\rho(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}d\mathbf{x}' + E^{XC}[\rho] \right) \quad (2.3.27)$$

$$= v(\mathbf{x}) + \frac{1}{2} \int \frac{\rho(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' + \frac{\delta E^{XC}[\rho]}{\delta \rho \mathbf{x}} \quad (2.3.28)$$

$$= v(\mathbf{x}) + v_H(\mathbf{x}) + v_{XC}(\mathbf{x}). \quad (2.3.29)$$

In the above none of the functional derivatives, except the one with respect to  $E^{XC}$ , have a dependence on  $\rho^{(n)}$  thus leaving equation (2.3.25) easily applied. Addressing the two last terms contributing to  $v_{KS}$  namely the Hartree potential and the exchange-correlation potential. Finding the Hartree potential corresponds to solving Poisson's equation known from electrostatics reads  $\nabla^2 v_h(\mathbf{x}) = -4\pi\rho(\mathbf{x})$ . In fact, Poisson's equation is the more fundamental definition of the Hartree potential, and the solutions depend on the space on which it is solved. For example, in flat 2D, the Coulomb potential (over which the Hartree potential is integrated) becomes  $\frac{1}{2\pi} \ln(|x - x'|)$  instead of  $\frac{-1}{4\pi|x-x'|}$ . In curved space, the solution will look different yet, arising from the Green's function to the Laplace-Beltrami operator. Fortunately, we do not have to solve it analytically, but compute it by solving Poisson's equation as a numerical matrix equation. Thus, everything in the above can be calculated exactly except the exchange-correlation potential. The problem of finding an appropriate approximation of the exchange-correlation potential, which is expected to be small correction is a much simpler problem than using appropriate functionals for  $F[\rho]$ . How the potential actually looks or whether it is even computable is an open question, but due to the small nature of the correction crude approximations can yield reasonable results. The simplest

implementations for exchange-correlation are Local Density Approximation(LDA) which is dealt with in section 3.3.2 as well as solving The Kohn-Sham equations which now reads

$$\left[ -\frac{\hbar^2}{2m} \nabla^2 + v(\mathbf{x}) + v_H(\mathbf{x}) + v_{XC}(\mathbf{x}) \right] \phi_i = \varepsilon_i \phi_i, \quad (2.3.30)$$

in an iterative fashion in section 3.3.3.

The first step towards reaching the goal of developing a DFT method on the manifold surfaces will be to create a solver for the relevant differential equations within DFT. For this a computational 2-dimensional FEM implementation is constructed. While FEM is a well studied and well-known method for PDEs it is rarely used in the setting of quantum mechanics. While the main FEM implementation is a standard approach it was build from scratch to allow for it to be tailored to the fullerene geometry but more importantly makes it possible to introduce curved space. This is necessary since the Laplace operator within a standard FEM assumes flat space, while its generalized counterpart, namely the Laplace-Beltrami operator, can account for curved spaces.

This section will first aim to give the reader an understanding of how the fullerene triangulation and matrix/vector assemblies in a standard FEM are constructed mathematically. Afterwards the necessary changes to account for curved space are presented. The attention is then turned to the computational implementation of the stated methods, where vital code snippets will be presented and explained.

Once the FEM software with flat and curved space is in place the attention is turned to density functional theory. Here the approach to the fully 2-dimensional DFT on non-euclidean manifolds that has been developed will be presented. This will include an approach to the exchange-correlation using a local density approximation and an introduction to the iterative self-consistent field(SCF) loop. With this in place the computational scheme of the aforementioned SCF loop is given.

To allow for some sort of verification of the FEM solver the heat equation is solved using forward Euler time integration. This verification is however not possible without a proper visualization of the solutions. An explanation is therefore needed of how a 2-dimensional visualization can be achieved. This visualization is achieved by mapping the polynomial representation of the FEM onto pixels. This, along side other important aspects of the visualization is given and will be used to show results on fullerene unfoldings in the Eisenstein plane in section 4.

Finally the construction of the sparse adjacency matrix in the dual representation of a fulleroid torus is given. This will be used to emphasize the adaptive nature of the PDE solver, interesting for a generalization towards fulleroids in the distant future.

### 3.1 FEM Implementation

The differential equations we wish to solve for DFT are the Kohn-Sham equations and Poisson's equations while the heat equation is used to validate and test the implementation. The necessary sub-routines for a computational implementation are in broad strokes:

1. Creating a mesh of the given fullerene.
2. Assembly of problem dependent matrices and vectors.
3. Solving the system of equations.
4. Presentation/Visualization of the solution.

The second task is where the standard flat space formulation will be extended to account for curvature. Thus following the above it seems only natural to start with the problem of meshing, which was presented in section 2.2.4.

### 3.1.1 Triangulation of Fullerenes

Keeping in mind that we wish to solve differential equation on fullerenes without any reference to the 3-dimensional space, then it makes sense to approach fullerenes purely in a geometrical sense, as hollow 3D structures defined by connecting 2D polygons, namely pentagons and hexagons which can be seen as a non-euclidean 2-dimensional surface. For a possible mesh one could choose to use the pentagons and hexagons as the cells and equip them with shape functions accordingly. However this implementation uses triangular shapes not only due to their simplicity, but also their adaptability to finer meshes, which will be explained shortly.

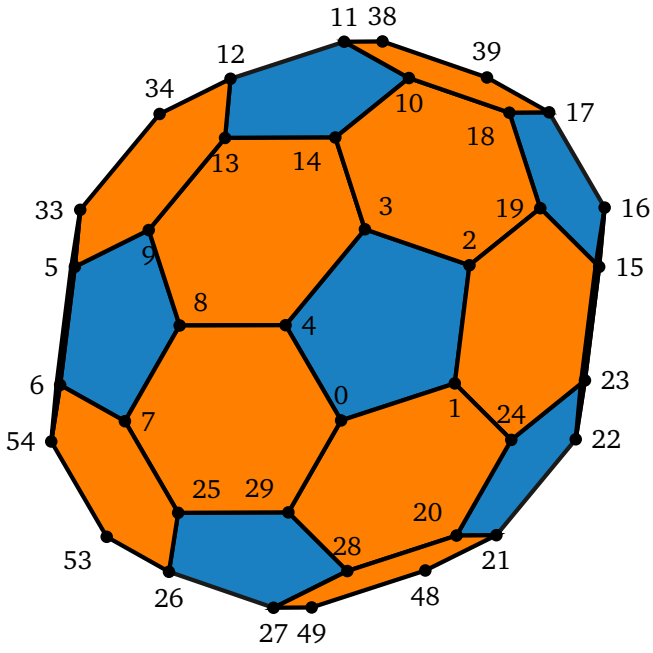
A pre-existing software courtesy of *James Emil Avery* outputs the sparse adjacency matrix introduced in section 2.1.2, with an example of vertex/atom labelling shown in figure 3.1 for the Buckyball. The software constructs two arrays where each row contain the labels defining a pentagon or hexagon. The pentagon and hexagon arrays will therefore be of size  $12 \times 5$  and  $N_h \times 6$  respectively, with  $N_h$  denoting the number of hexagons. Each vertex will have three neighbouring vertices and each label point will appear exactly three times in the arrays produced.

One type of triangulation performed is done by placing a new vertex with a numbered label at the middle of each polygon. This will create a mesh consisting of 5 isosceles triangle for each pentagon, and 6 equilateral triangles for the hexagons. Thus the total number of vertices  $N_v$  for triangulation is  $N_v = N + N_p + N_h$ , where  $N$  is the number of atoms/vertices in the original structure and  $N_p$  is the number of pentagons. Computation of this triangulation on the  $C_{60}I_h$  molecule is illustrated in figure 3.2 where the gray dots represent the added vertices.

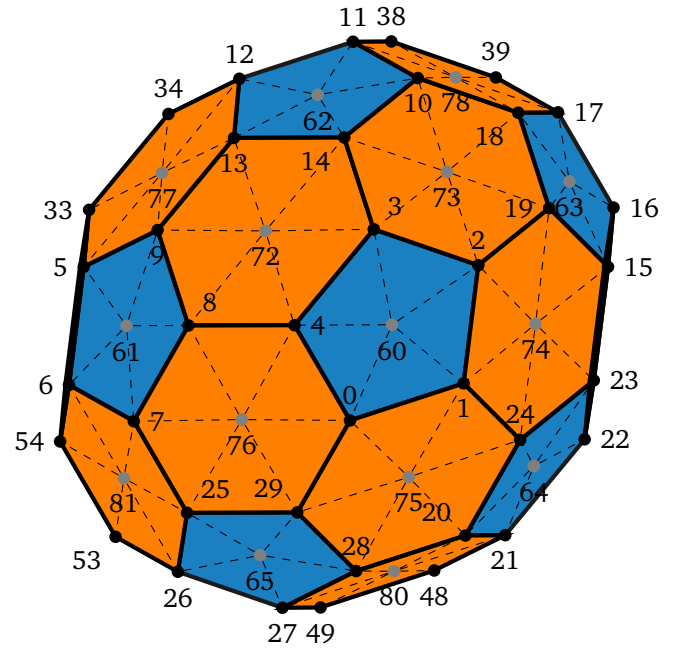
From the triangles formed by the dashed and full lines in figure 3.2 a corresponding neighbouring array for all the triangles is computed. It has been designed such that the first  $5N_p$  rows contain all the isosceles triangles. The fixed number of 12 pentagons present in a fullerene predetermines the number of columns necessary in the triangulation. The number of hexagons present can be determined by  $N = (5N_p + 6N_h)/3$  which are all atoms present in each face divided by the number of bonds for each atom. Continuing from this expression  $N = (5 \cdot 12 + 6N_h)/3 = 20 + 2N_h \Leftrightarrow N_h = N/2 - 10$  follows. This helps us define the number of faces  $N_f = N_p + N_h = N/2 + 2$ . The triangulation is a fairly simple piece of code which can be found in script `triangulation.py` in the stated Github repository. For figure 3.2 the sparse adjacency matrix for the triangulation would read

$$T = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ 1 & 2 & 3 & 4 & 0 & 6 & \dots \\ 60 & 60 & 60 & 60 & 60 & 61 & \dots \end{pmatrix}^T \quad (3.1.1)$$

An alternative easily constructed triangulation for the fullerenes is the already discussed dual representation. The dual yields a mesh of  $N_T = N$  triangles, while the former representation has  $N_T = N + N_f = 5N_p + 6N_h$ . The FEM software written has both triangulations implemented, due to both of them having advantages and it being uncertain which will feature in the final implementation, since the approach to triangulation to be used in the future is still to be determined. The data files received for the FEM implementation already have the dual sparse adjacency matrix given, with the first 12 rows defining pentagon faces.



**Figure 3.1.:** The  $C_{60} - I_h$  molecule with pre-computed labels for each atom with red and orange illustrating pentagons and hexagons respectively.



**Figure 3.2.:** Adding vertex the center of each polygon, here shown in gray, yields the triangulation here performed on the  $C_{60}I_h$  molecule.

### 3.1.2 FEM Matrices and Vector Assembly

With the triangulation in place we take a look at the necessary matrices and vectors for solving the Kohn-Sham equations and Poisson's equation which read

$$\left[ -\frac{\hbar^2}{2m} \nabla^2 + v_{eff}(\mathbf{x}) \right] \psi_\alpha(\mathbf{x}) = \varepsilon_\alpha \psi_\alpha(\mathbf{x}) \quad \text{and} \quad \nabla^2 v_h(\mathbf{x}) = -4\pi\rho(\mathbf{x}). \quad (3.1.2)$$

Expressing the above in weak formulation is straightforward considering section 2.2.2, however the argument that has the boundary integral be zero in the stiffness matrix is now due to the fact that this system has no boundary. The equations in matrix form then read

$$\left( \frac{1}{2} \mathbf{L} + \mathbf{V} \right) \psi_\alpha(\mathbf{x}) = \varepsilon_\alpha \mathbf{M} \psi_\alpha(\mathbf{x}) \quad \text{and} \quad \mathbf{L} v_h = \mathbf{f} \quad (3.1.3)$$

where

$$L_{ij} = \int_{\Omega} \nabla \varphi_i(\mathbf{x}) \cdot \nabla \varphi_j(\mathbf{x}) d\mathbf{x}, \quad V_{ij} = \int_{\Omega} v_{eff}(\mathbf{x}) \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x}, \quad (3.1.4)$$

$$M_{ij} = \int_{\Omega} \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x} \quad \text{and} \quad f_i = - \int_{\Omega} 4\pi\rho(\mathbf{x}) \varphi_i(\mathbf{x}) d\mathbf{x}. \quad (3.1.5)$$

The matrices and vectors required for solving the PDE of the heat equation is included in the above. How this is approached will be dealt with in section 3.4.

All the integrals in the above matrices are over the entire region  $\Omega$ , which can be written as the sum of the integrals taken over the individual triangles  $K$ , e.g:

$$L_{ij} = \int_{\Omega} \nabla\varphi_i(\mathbf{x}) \cdot \nabla\varphi_j(\mathbf{x})d\mathbf{x} = \sum_K \int_K \nabla\varphi_i(\mathbf{x}) \cdot \nabla\varphi_j(\mathbf{x})d\mathbf{x} = \sum_K L_{\hat{i}\hat{j}}^K, \quad (3.1.6)$$

where  $L_{\hat{i}\hat{j}}^K$  is the local matrix for the  $K$ 'th triangle of size  $n_{dof} \times n_{dof}$ . In practice both  $M$  and  $L$  are calculated by computing the local matrices and adding contribution from each triangle to the global system. Here the indices transform locally from  $\hat{i}\hat{j}$  to  $ij$  through the global nodes defining the triangle  $K$ . Note that if the system was constructed by geometrically identical triangles, all local matrices would be identical, and the entire assembly could be performed with the  $n_{dof} \times n_{dof}$  matrix. This approach is obviously possible for the dual representation of equilateral triangles. But the alternative triangulation described for the system will due to the two geometrically different triangles of the computed triangulation need two local matrices to compute  $M$  and  $L$ . Assembly of  $f$  and  $V$  are slightly different on account of the integrals not only being shape function dependent. This will be addressed along with relevant code snippets in section 3.2.3 and 3.2.2.

### 3.1.3 The Reference Element

In a FEM solver the need for evaluating the integrals on a complicated mesh consisting of many geometrically different triangles, quickly becomes quite complicated. The universal approach is to define a unit cell also referred to as the reference element. The idea is to transform all the triangles to a geometrical simple cell equipped with shape functions and then evaluate the integrals on said cell. This work used a right angled triangle with corners in  $\{0,0\}$ ,  $\{0,1\}$  and  $\{1,0\}$  and the variables  $\{\xi, \eta\}$  to denote the unit cell. To avoid confusion, FEM literature usually use  $\{x, y\}$  when describing physical elements(triangles in the mesh) and  $(\xi, \eta)$  for the reference element.

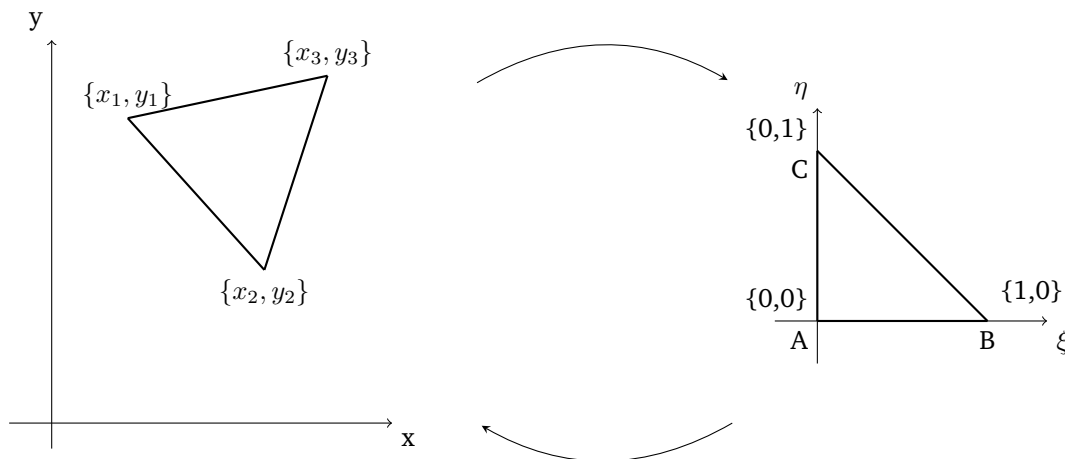


Figure 3.3.: The triangle transformation from a physical element  $\{x,y\}$  to the unit cell  $\{\xi,\eta\}$ .

The coordinate transformation seen in figure 3.3 from the  $\{x, y\}$  variables of the physical triangles, to the  $\{\xi,\eta\}$  variables of the unit cell is performed by

$$x = x_1 + (x_2 - x_1)\xi + (x_3 - x_1)\eta \quad (3.1.7)$$

$$y = y_1 + (y_2 - y_1)\xi + (y_3 - y_1)\eta, \quad (3.1.8)$$



where the triangle notation seen in figure 3.3 is used. Using the above will have a Jacobian matrix for this transformation of

$$\mathbf{J} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}. \quad (3.1.9)$$

The transformations back and forth are then easily conveyed to a matrix representation using  $\mathbf{J}$ , and takes the form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \mathbf{J} \begin{pmatrix} \xi \\ \eta \end{pmatrix} \quad \text{and} \quad (3.1.10)$$

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \mathbf{J}^{-1} \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}. \quad (3.1.11)$$

It is now possible with any coordinate set within the physical element  $\{x, y\}$  to calculate the value of a basis function. This is done by transforming coordinates into the unit cell  $\{\xi, \eta\}$  and passing the new coordinates to the specific basis function, which is defined on the unit cell e.g. for  $M_{ij}^K$

$$M_{ij}^K = \int \int_K \varphi_i(x, y) \varphi_j(x, y) \, dx dy = |\mathbf{J}_K| \int \int_k \varphi_i(\xi, \eta) \varphi_j(\xi, \eta) \, d\xi d\eta. \quad (3.1.12)$$

Where the determinant of the Jacobian in the case of triangular transformations is equal to two times the area of the original shape. The shape functions defined on the reference element are for the linear functions

$$\varphi_1 = 1 - \xi - \eta, \quad \varphi_2 = \xi, \quad \varphi_3 = \eta, \quad (3.1.13)$$

and for the quadratic

$$\varphi_4 = 2\left(1 - \xi - \eta\right)\left(\frac{1}{2} - \xi - \eta\right), \quad \varphi_5 = 2\left(\xi - \frac{1}{2}\right)\xi, \quad \varphi_6 = 2\left(\eta - \frac{1}{2}\right)\eta, \quad (3.1.14)$$

$$\varphi_4 = 4\xi(1 - \xi - \eta), \quad \varphi_5 = 4\xi\eta, \quad \varphi_6 = 4\eta(1 - \xi - \eta). \quad (3.1.15)$$

With all of the above illustrated in figure 3.4.

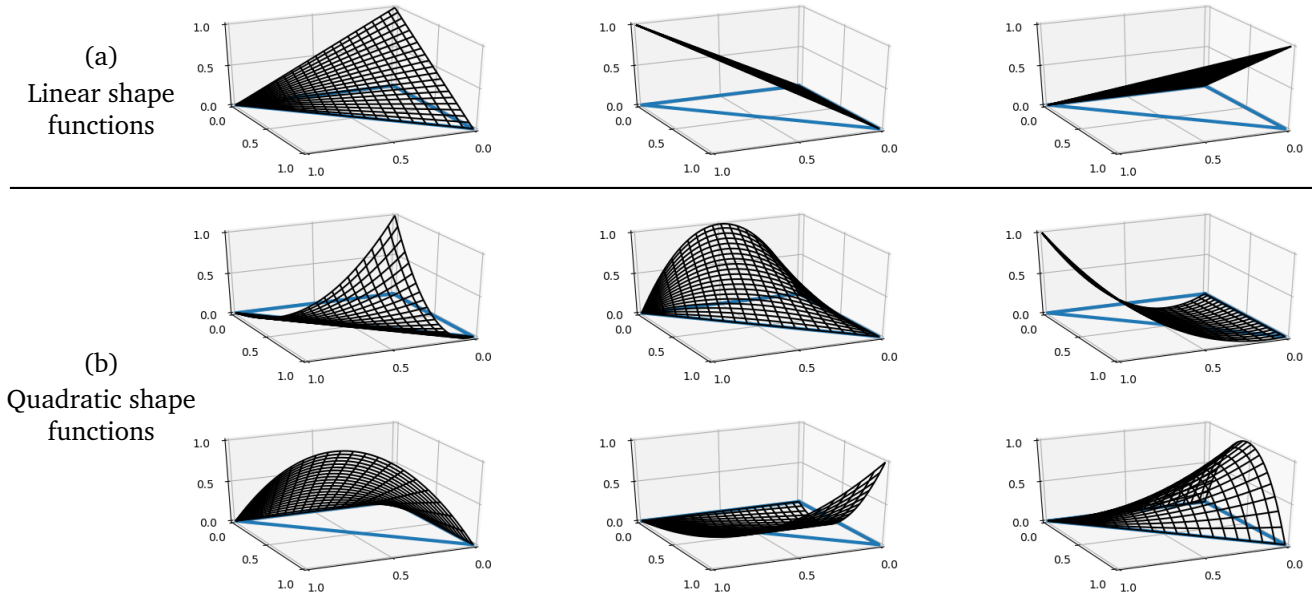
Now to calculate  $\mathbf{L}$  in a similar method, evaluation of gradients of shape functions is needed. The simple expressions for the shape functions in the transformed coordinates in equations (3.1.13), (3.1.14) and (3.1.15) yield easily obtained gradients with respect to the  $\{\xi, \eta\}$  variables, and applying the chain rule aids in expressing the gradients in the physical variables. Introducing the notation

$$\nabla^t \varphi_i = \left( \frac{\partial \varphi_i}{\partial \xi} \quad \frac{\partial \varphi_i}{\partial \eta} \right)^T, \quad (3.1.16)$$

where  $\nabla^t$  represents the gradient of basis functions  $\varphi_i$  with respect to  $\xi$  and  $\eta$ . The derivatives above can be calculated using the chain rule for dependent variables,

$$\frac{\partial \varphi_i}{\partial \xi} = \frac{\partial \varphi_i}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial \varphi_i}{\partial y} \frac{\partial y}{\partial \xi}, \quad (3.1.17)$$

$$\frac{\partial \varphi_i}{\partial \eta} = \frac{\partial \varphi_i}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial \varphi_i}{\partial y} \frac{\partial y}{\partial \eta}, \quad (3.1.18)$$



**Figure 3.4.:** The implemented basis functions where (a) visualize the linear shape functions on the reference cell outlined with blue and (b) visualize the quadratic shape functions.

which can be written in matrix form as

$$\begin{pmatrix} \frac{\partial \varphi_i}{\partial \xi} \\ \frac{\partial \varphi_i}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix} \begin{pmatrix} \frac{\partial \varphi_i}{\partial x} \\ \frac{\partial \varphi_i}{\partial y} \end{pmatrix}, \quad (3.1.19)$$

By recognizing the square matrix as the transpose of the Jacobian in eq. (3.1.9) the above can alternatively be expressed as

$$\nabla^t \varphi_i = \mathbf{J}^T \nabla \varphi_i, \quad (3.1.20)$$

from which the expression for the gradient in the original system follow as

$$\nabla \varphi_i = (\mathbf{J}^T)^{-1} \nabla^t \varphi_i. \quad (3.1.21)$$

Constructing a FEM is simplified with the use of this unit method, since shape functions are defined in  $\{\xi, \eta\}$  and therefore independent of the mesh. Only the computation of the Jacobian in equation (3.1.9) of geometrically varying finite elements in the triangulation is needed.  $W_{ij}^K$  can then be evaluated through the transformation by

$$W_{ij}^K = \int \int_K \nabla \varphi_i(x, y) \cdot \nabla \varphi_j(x, y) \, dx dy \quad (3.1.22)$$

$$= |\mathbf{J}_K| \int \int_k ((\mathbf{J}^T)^{-1} \nabla^t \varphi_i(\xi, \eta)) \cdot ((\mathbf{J}^T)^{-1} \nabla^t \varphi_j(\xi, \eta)) \, d\xi d\eta. \quad (3.1.23)$$

Now that  $M$  and  $W$  are expressions based on the reference element let us take a look at how to actually evaluate the integrals.

### 3.1.4 Integral Evaluation with Gaussian Quadrature

The polynomial integrals on the triangles are solved by using Gaussian quadrature. Gaussian quadrature is a numerical tool for solving definite integrals of polynomial functions exactly. The integrals are solved taking a sum over the polynomial function values  $f(\mathbf{x})$  at specially chosen coordinates  $(\mathbf{x}_i)$  each associated with a weight  $w_i$  i.e.

$$\int_R f(\mathbf{x}) d^n \mathbf{x} = \sum_{i=1}^{n_q} w_i f(\mathbf{x}_i), \quad (3.1.24)$$

over some  $n$ -dimensional domain  $R$ . The number of quadrature points limits the polynomial order  $f(\mathbf{x})$  can take if the integral evaluation is to be exact. While Gaussian quadrature evaluate integrals exactly for polynomial functions, quadratures can be implemented for many classes of functions. The  $\mathbb{R}^2$  region we are concerned with is an equilateral triangle unitcell. A  $n$ -point 1-dimensional quadrature rule can solve polynomials of orders up to  $2n - 1$ . The possible order when dealing with a two-dimensional square element constructed through tensor products of the one-dimensional element can be found to be  $(2n - 1)^2$ . However in the 2-dimensional case of a non rectangular shape no general statement for correspondence between quadrature points and polynomial order exists.

The work done in [16] produces quadrature points with corresponding weights for a 7, 25, 54, 85, 126 and 175 point implementation on an equilateral triangle as in figure 3.5. This can solve polynomials of order 5, 10, 15, 20, 25 and 30 respectively. Here the polynomial order for a function  $f(x, y)$  is defined to be the product with the highest sum of exponents e.g.  $(x^2y^3 + xy)$  consists of two terms with an order of 5 and 2 given the polynomial an order of 5. Note that the weights must be normalized to the area of the equilateral triangle which in case of figure 3.5 means  $\sum_{i=1}^{n_q} w_i = \sqrt{27}/4$ . The 7-point and 54 point example seen in figure 3.5 are both implemented.

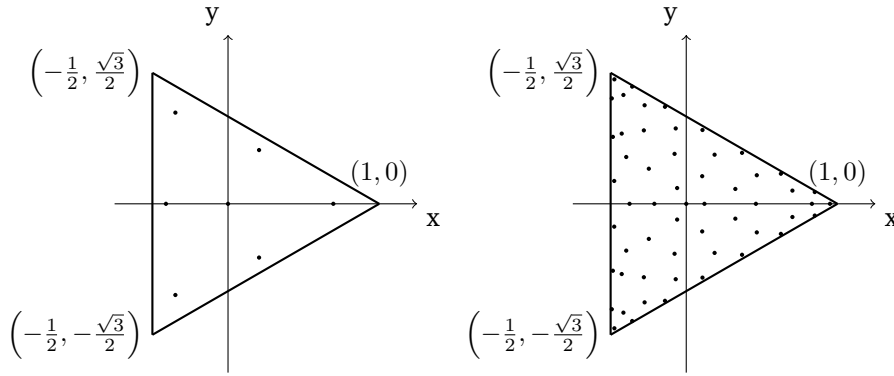
The polynomial degree of the shape functions we wish to integrate will fully decide the necessary number of quadrature points. Several of the FEM matrices are integrals of products between shape functions, e.g. take the matrix  $V_{ij}$  as an example

$$V_{ij} = \int_{\Omega} v_{eff}(\mathbf{x}) \varphi_i(\mathbf{x}) \varphi_j(\mathbf{x}) d\mathbf{x}. \quad (3.1.25)$$

Using quadratic shape functions will yield a polynomial order of 4 for the product of the two basis functions, while the effective potential  $v_{eff}(\mathbf{x})$  is a second order piece-wise polynomial function as well leaving a solution requirement of 6th polynomial order. The 7 point implementation with an solvable order of 5 is thus insufficient. The quadrature implementation have been tested thoroughly by comparing integral results from Wolfram Mathematica[17]. This exposed mistakes in the resulting 25-point coordinates and/or weights from [16]. It simply did not compute the correct results using identical methods for all  $n$ -point implementations.

### 3.1.5 Stiffness Matrix in Curved Space

The Laplacian/stiffness matrix that has been introduced and constructed is applicable in Euclidean flat spaces. However the bumpy manifold surfaces in question can not be equipped with a global coordinate system and are inherently non-Euclidean. The Laplacian must account for this to simulate the physical system. The Laplace operator is in fact a special case of the more intricate Laplace-Beltrami operator,



**Figure 3.5.:** The 7-point and 54-point quadrature which can respectively compute definite integrals of any polynomial function of order 5 and 15.

known from the mathematical field of differential geometry. The continuous Laplace-Beltrami operator for a  $n$ -dimensional space reads

$$\nabla_{LB}^2 = \sum_{i=1}^d \sum_{j=1}^d \frac{1}{\sqrt{|g|}} \frac{\partial}{\partial x^i} \sqrt{|g|} g^{ij} \frac{\partial}{\partial x^j}, \quad (3.1.26)$$

with contravariant metric tensor  $g^{ij}$  and Jacobian determinant  $\sqrt{|g|}$ . This rather complicated expression will need to be expressed in a discrete sense to be implemented. This is no straightforward task and is the subject to ongoing research in mathematics and computer science. Some important considerations are locality of the discrete operator and convergence of solutions when further refining a mesh representing a smooth surface.

A popular discrete form of the Laplace-Beltrami operator for a triangular mesh, is the cotangent Laplacian. This operator has been derived in various contexts such as mean curvature flow [18]. This uses the cotangent of the inner angles in the triangular mesh. This discrete operator for a triangular mesh can be applied to systems with function values discretely defined at the triangle vertices, and will therefore be limited to the linear basis implementation.

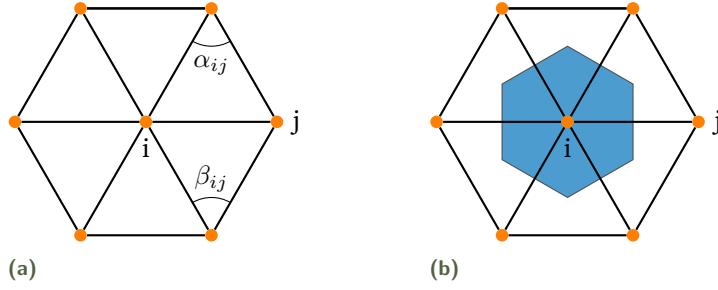
The cotan Laplacian  $L^c$  is defined based on adjacent triangles. Each entry  $L_{ij}^c$  will be associated with a weight as

$$L_{ij}^c = \begin{cases} \sum_j w_{ij}, & \text{if } i = j. \\ -w_{ij}, & \text{if } (i, j) \in \mathcal{E}. \\ 0, & \text{if } (i, j) \notin \mathcal{E}. \end{cases} \quad (3.1.27)$$

where  $\sum_j w_{ij}$  is the sum over all non-zero weights i.e.  $(i, j) \in \mathcal{E}$ . An edge  $(i, j)$  in the graph representation of the dual is shared by two triangles. The two angles in the triangles not located at  $i$  or  $j$  as seen in figure 3.6a are used to define the weights. An early simple approach to these weights in the cotangent Laplacian reads

$$w(i, j) = \frac{1}{2} (\cot(\alpha_{ij}) + \cot(\beta_{ij})). \quad (3.1.28)$$

While this approach depends on the angles constructing the triangular mesh it contains no information of scaling whatsoever. As long as all elements in the mesh are scaled accordingly it yields the exact same matrix. There are therefore several approaches to refine the discretization by using an area dependent normalization factor. We will be concerned with a normalization with respect to each vertex



**Figure 3.6.:** (a) The angles  $\alpha_{ij}$  and  $\beta_{ij}$  associated to the matrix entry  $W_{ij}^c$  and used to calculate the weight  $w(i, j)$ . (b) The Voronoi element area for the index  $i$  constructed of middle of each triangle, which for our triangulation creates the pentagons and hexagons.

associated Voronoi element as in [19]. A Voronoi diagram is the result of partitioning a plane into convex polygons. Each polygon will contain exactly one of the original discrete points, with a polygonal edge being orthogonal to the vector between the neighbouring points. In a square lattice for discrete points this will yield a Voronoi diagram of adjacent squares of equal size. In our particular case the Voronoi element of the dual will yield the penta- and hexagonal shapes present in the fullerene. This is illustrated for a hexagon in figure 3.6b. Introducing the normalization in equation (3.1.28) looks like so

$$w(i, j) = \frac{1}{2A_i^v} (\cot(\alpha_{ij}) + \cot(\beta_{ij})), \quad (3.1.29)$$

here the  $A_i^v$  refer to the area of the Voronoi element for vertex  $i$  illustrated in figure 3.6b. Since the Voronoi element reconstruct the fullerene polygons we only need to be concerned with two different areas.

It is important to note that there are various approaches to this normalization of the cotan Laplacian. Examples of this is a normalization depending on a third of the total area of all adjacent triangles  $A_i/3^1$  or alternatively  $1/\sqrt{A_i^v A_j^v}$  as presented in [20]. The cotangent Laplacian matrix is quite simple to construct, especially for the dual triangulation where all angles are  $\pi/3$  for which

$$\cot(\pi/3) = \frac{\sin(\pi/3)}{\cos(\pi/3)} = \frac{\frac{1}{2}}{\frac{\sqrt{3}}{2}} = \frac{1}{\sqrt{3}}. \quad (3.1.30)$$

While perfectly applicable to our mesh the operator can not be applied to a mesh containing blunt triangles which yields negative weights due to the sinusoidal term.

A convenient way to construct the matrix in question will be through the matrix multiplication

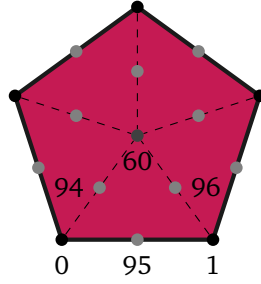
$$\mathbf{L}^c = \mathbf{A}^{-1} \mathbf{L}^{cot}. \quad (3.1.31)$$

Here  $\mathbf{A}$  is a diagonal matrix defining all Voronoi areas for the vertices and  $\mathbf{L}^{cot}$  has entries of all relevant cotangent values, both of size  $N_{dof} \times N_{dof}$ . This will allow for flexible implementation for easy problem reformulation. An example could be Poisson's equation which will read

$$\mathbf{L}^c \mathbf{u} = \mathbf{f} \Rightarrow \mathbf{A}^{-1} \mathbf{L}^{cot} \mathbf{u} = \mathbf{f}. \quad (3.1.32)$$

Here the left hand side is not symmetric, which can be advantageous when solving linear systems of equations. Among such advantages is the applicability of the Cholesky decomposition which speed up

<sup>1</sup>This is the same as the Voronoi element for our dual mesh consisting of equilateral triangles, this is however not generally the case.



**Figure 3.7.:** All nodes present in a pentagon with quadratic shape function. The labels from  $GlobalMapping(T)$  for  $C_{60} - I_h$  molecule are shown for one triangle. The dark dots are the original vertices/atoms, the dark gray came when performing the triangulation while the light gray dots are the added nodes for the quadratic shape functions.

solutions for symmetric positive-definite and semi definite matrices greatly. This is mostly a computational aspect but worth consideration. Constructing the matrices separately now allow for us to make the left hand side symmetric simply through matrix multiplication as

$$\mathbf{L}^{cot} \mathbf{u} = \mathbf{A} \mathbf{f}. \quad (3.1.33)$$

The function call to construct the matrices in question reads will be presented in listing 3.9.

## 3.2 FEM Modules

The FEM software written consists mainly of three modules namely the Basis, UnitCell and Assemble modules. They will be introduced accordingly, with python code snippets for the vital computations. For an overview of the current directory overview the reader is referred to appendix B. For a concise description the code presented will be for an input of the dual and often shortened omitting import statements etc. Comments will be given when the extension to the alternative triangulation is not trivial. The following three modules are the bedrock of FEM solver suited to the fullerene graph representation. In section 3.1.5 the introduction of the discretize approach to the Laplace-Beltrami operator was given. The program has been neatly constructed such that the extension to include the curved space operator of the cotangent Laplacian can enter without any restructuring.

### 3.2.1 Basis Modules

The complexity of the matrix/vector assembly in an FEM vary with the order of the implemented basis functions. The implementation as we will see in the coming sections takes a module named *Basis* as an input. This module consists of 5 functions relevant for the polynomial order of the basis functions. In this work the quadratic and linear modules *BasisFunctions\_Quadratic* and *BasisFunctions\_Linear* have been written and can be passed to the software. Two of the functions *LocalDOF()* and *GlobalDOF(T)* return the local and global degrees of freedom which for the latter depend on the triangulation  $T$ . *BasisFunctionValue(i, point)* and *BasisGradientValue(i, point)* return the function and gradient value of a specific point in the reference element using equations (3.1.13), (3.1.14) and (3.1.15), where  $i$  is an integer referring to a specific shape function. The last function *GlobalMapping(T)* returns a neighbouring array somewhat like the triangulation, but containing all  $n_{dof}$  neighbouring nodes in each triangle. This will be referred to at the global map of the system. Taking the quadratic implementation as an example, which as introduced in section 2.2.4, has 6 nodes per triangle. Here each vertex node is shared by either

5 or 6 other triangles, while the mid-point nodes is common to 2 triangles, illustrated in figure 3.7. Incorporating a new module with implemented basis functions of polynomial order  $n$  is a task of coding these five functions.

### 3.2.2 UnitCell Module

The UnitCell.py module incorporates all the computations done on the unit cell, this therefore mainly deals with the local integral assembly. Evaluating  $M_{ij}^K$  using quadratures as in equation (3.1.12) yields

$$M_{ij}^K = |\mathbf{J}_K| \sum_{q=1}^{n_q} \varphi_i(\xi_q, \eta_q) \varphi_j(\xi_q, \eta_q) w_q. \quad (3.2.1)$$

The computation of this local matrix is performed by calling the function *LocalOverlap* in the UnitCell module which reads

```

1 def LocalOverlap(x_q, w_q, Basis, determinant):
2     local_dof = Basis.LocalDOF()
3     M_local = np.zeros([local_dof, local_dof])
4     for i in range(local_dof):
5         for j in range(local_dof):
6             for q in range(x_q.shape[0]):
7                 #Looping through all quadrature points and weights
8                 point = CoordinateTransformation(x_q[q,:])
9                 M_local[i,j] += Basis.BasisFunctionValue(i, point) *\
10                    Basis.BasisFunctionValue(j, point) *\
11                    w_q[q] * determinant
12     return M_local

```

Listing 3.1: LocalOverlap function for computing the local mass matrix  $M^K$ .

By passing quadrature coordinates  $x_q$ , quadrature weights  $w_q$ , the relevant basis module and the determinant of the Jacobian for the transformation in listing 3.1, obtaining the local matrix is somewhat straight forward. The function loop over all entries in the local  $n_{dof} \times n_{dof}$  matrix and then transform a quadrature point from the equilateral element to the right angled element in line 9 through the function *CoordinateTransformation*. Once the transformed coordinate is obtained each quadrature contribution is added together to the entry matrix  $[i, j]$ . Note that the weights are normalized to  $\frac{1}{2}$  for the area of the right angled triangle, this is done prior to calling the function above. *CoordinateTransformation* used above is a implementation of the coordinate transformation given in equation (3.1.11) and reads

```

1 def CoordinateTransformation(coordinate):
2     from FEM.shape_construction import FE_construction
3     Element = FE_construction('triangle_equil')[1]
4     Jacobian=np.array([[Element[1,0]-Element[0,0],Element[2,0]-Element[0,0]],
5                       [Element[1,1]-Element[0,1],Element[2,1]-Element[0,1]]])
6     inv_Jacobian = np.linalg.inv(Jacobian)
7     quad_reference = np.dot(inv_Jacobian, coordinate - Element[0,:])
8     #quadrature coordinates in the reference element
9     return quad_reference

```

Listing 3.2: Coordinate transformation from physical element to reference element.

In the above *FE\_construction* called in line 3 is a simple module which computes and returns the determinants of the Jacobian and coordinates defined in the array *Element* making up the various triangular elements used. This function is used for transforming the quadrature points and therefore only uses the equilateral element called in line 3 no matter what triangulation is used.

$L_{ij}^K$  is obtained in a similar fashion to  $M_{ij}^K$ , where by looping through all entries in a function named *LocalGradientOverlap* and evaluating in all the relevant quadrature points.

```

1 def LocalGradientOverlap(x_q, w_q, Basis, determinant, Element):
2     local_dof = Basis.LocalDOF()
3     L_local = np.zeros([local_dof, local_dof])
4     for i in range(local_dof):
5         for j in range(local_dof):
6             for q in range(x_q.shape[0]):
7                 grad_basis_i = Gradient(i, x_q[q,:], Basis, Element)
8                 grad_basis_j = Gradient(j, x_q[q,:], Basis, Element)
9                 L_local[i,j] += np.dot(grad_basis_i, grad_basis_j) *\
10                                w_q[q] * determinant
11     return L_local

```

Listing 3.3: The function *LocalGradientOverlap* which calculate the local stiffness matrix  $\mathbf{W}^K$ .

To construct the local matrix a function named *Gradient*, used in line 9 and 10, returns the unit cell gradient of basis function  $\hat{i}$  for a quadrature point, it reads

```

1 def Gradient(n, coor, Basis, Element):
2     quad_reference = CoordinateTransformation(coor)
3     gradient_reference = Basis.BasisFunctionGradient(n, quad_reference)
4     Jacobian=np.array([[Element[1,0]-Element[0,0],Element[2,0]-Element[0,0]],
5                       [Element[1,1]-Element[0,1],Element[2,1]-Element[0,1]]])
6     trans_Jacobian = np.transpose(Jacobian)
7     gradient_physical = np.linalg.solve(trans_Jacobian, gradient_reference)
8     return gradient_physical

```

Listing 3.4: The function *Gradient* which compute the gradient for basis function n in the physical space when given coordinates

Starting by once again transforming the coordinate to the reference system with *CoordinateTransformation* then the gradient in the reference cell can found by calling *BasisFunctionGradient* from the input basis module. We can then solve for the gradient in the physical system using equation 3.1.20. This is executed in line 7 with the transpose of the Jacobian and the gradient in the reference system. The local matrix assembly in line 11 in listing 3.3 is then computed similarly to listing 3.1<sup>2</sup>.

Turning the attention to  $\mathbf{f}$  and  $\mathbf{V}$ . The integrals associated with both of these does not solely depend on the basis functions. Here the density and effective potential which are globally varying functions enter in entries  $f_i$  and  $V_i$  respectively. These can therefore not be completely constructed from a local

<sup>2</sup>The local matrices assembled have been thoroughly tested and compared to Mathematica calculation for several types of triangles.



matrix/vector as in the case of  $M$  and  $L$ . Writing up the integral computations with quadratures and sum over relevant triangles  $K$  looks like so

$$V_{ij} = \sum_K \sum_{q=1}^{n_q} v_{eff}(x_q, y_q) \varphi_i(x_q, y_q) \varphi_j(x_q, y_q) w_q \quad (3.2.2)$$

$$f_i = - \sum_K \sum_{q=1}^{n_q} 4\pi \rho(x_q, y_q) \varphi_i(x_q, y_q) w_q. \quad (3.2.3)$$

We can not perform matrix assembly of exact integrals on the unit cell the, nevertheless we can return locally dependent matrices, which can aid in the global construction. Using  $(\mathbf{x}_q) = (x_q, y_q)$  for notational simplicity the local matrix in question for  $V$  is of the form

$$\mathbf{V}_{\text{local}}^K = |\mathbf{J}^K| \begin{pmatrix} \varphi_1(\mathbf{x}_1)\varphi_1(\mathbf{x}_1)w_1 & \varphi_1(\mathbf{x}_1)\varphi_2(\mathbf{x}_1)w_1 & \dots & \varphi_1(\mathbf{x}_1)\varphi_{n_{dof}}(\mathbf{x}_1)w_1 \\ \varphi_2(\mathbf{x}_1)\varphi_1(\mathbf{x}_1)w_1 & \varphi_2(\mathbf{x}_1)\varphi_2(\mathbf{x}_1)w_1 & \dots & \varphi_2(\mathbf{x}_1)\varphi_{n_{dof}}(\mathbf{x}_1)w_1 \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{n_{dof}}(\mathbf{x}_1)\varphi_1(\mathbf{x}_1)w_1 & \varphi_{n_{dof}}(\mathbf{x}_1)\varphi_2(\mathbf{x}_1)w_1 & \dots & \varphi_{n_{dof}}(\mathbf{x}_1)\varphi_{n_{dof}}(\mathbf{x}_1)w_1 \\ & \varphi_1(\mathbf{x}_2)\varphi_1(\mathbf{x}_2)w_2 & \dots & \varphi_1(\mathbf{x}_{n_q})\varphi_{n_{dof}}(\mathbf{x}_{n_q})w_{n_q} \\ & \varphi_2(\mathbf{x}_2)\varphi_1(\mathbf{x}_2)w_2 & \dots & \varphi_2(\mathbf{x}_{n_q})\varphi_{n_{dof}}(\mathbf{x}_{n_q})w_{n_q} \\ & \vdots & \ddots & \vdots \\ \varphi_{n_{dof}}(\mathbf{x}_2)\varphi_1(\mathbf{x}_2)w_2 & \dots & \varphi_{n_{dof}}(\mathbf{x}_{n_q})\varphi_{n_{dof}}(\mathbf{x}_{n_q})w_{n_q} \end{pmatrix} \quad (3.2.4)$$

The above is of size  $n_{dof} \times (n_{dof} \cdot n_q)$  and contains quadrature evaluations for all  $n_q$  for every combination of the  $n_{dof}$  shape functions. Take the top row as an example we see the first  $n_{dof}$  entries are all combination of  $\varphi_1 \varphi_i$  evaluated in the first quadrature point with the corresponding weight. This is then repeated for the second, third, fourth etc. quadrature point. The row will therefore be of length  $n_{dof} \times n_{dof}$ .  $\mathbf{f}_{\text{local}}$  is similarly constructed from all combinations of  $\varphi_i(x_q, y_q)w_q$ . The last functions in the UnitCell module deal with the return of exactly these matrices that are used for the global construction. Constructing of  $\mathbf{V}_{\text{local}}^K$  is handled by the function *LocalPot* and reads

```

1 def LocalPot(x_q, w_q, Basis, determinant):
2     local_dof = Basis.LocalDOF()
3     V_local_quad = np.zeros([local_dof, local_dof * len(w_q)])
4     for q in range(len(w_q)):
5         point = CoordinateTransformation(x_q[q, :])
6         for i in range(local_dof):
7             for j in range(local_dof):
8                 V_local_quad[i, j + local_dof * q] = Basis.BasisFunctionValue(i, point) * \
9                                                         Basis.BasisFunctionValue(j, point) * \
10                                                         w_q[q] * determinant
11     return pot_local_quad

```

**Listing 3.5:** The *LocalPot* function which return the locally dependent matrix  $\mathbf{V}_{\text{local}}$ .

The first for loop initiated in line 4 with  $q$  is used to access the quadrature points and weights. For each iteration a  $n_{dof} \times n_{dof}$  matrix is added to the total matrix  $V\_local\_quad$  initiated in line 3. This contains all combinations of  $\varphi_i \varphi_j$  evaluated in quadrature point entry  $q$ . Computing  $\mathbf{f}$  is just a simplified version of the above and handled by the function *LocalLoadVectorMatrix*.

### 3.2.3 FEM\_Assembly Module

Before diving into the entire global assemblies a short but quite important function should be introduced. In equation (3.2.2) and (3.2.3) it was not discussed that the input of the effective potential and the density are evaluated in the quadrature coordinates, even though they are both vectors only with defined values at the nodes. However as we know the order of the basis functions also decide the polynomial approximation of these quantities. Then with the values at each node in a specific triangle a polynomial function value at any coordinate within the triangle can be obtained. This is done by taking the sum over the shape function value at the coordinate multiplied by the value at the corresponding node, which is computed in the function *continuous\_rep*

```
1 def continuous_rep(u, indices, Basis, point):
2     local_dof = Basis.LocalDOF()
3     val_polynomial = 0
4     import FEM.UnitCell_Computations.UnitCell as UnitCell
5     #reference point
6     point_ref = UnitCell.CoordinateTransformation(point)
7     for N in range(local_dof):
8         val_polynomial += u[indices[N]]*Basis.BasisFunctionValue(N, point_ref)
9     return val_polynomial
```

Listing 3.6: Function *continuous\_rep* which for a coordinate set within a triangle calculate the polynomial representation of the function using the function values at the triangle nodes.

Here, the input *u* is the function we wish to approximate in the finite element basis, while *indices* refer to the nodes and are the relevant entries into the vector *u*.

The FEM\_Assembly module control the global matrix assemblies which is based upon the input equation type, one wish to solve. The function *Assemble* will initialize all necessary computations given an input string 'equation\_type' and return the relevant matrices and/or vectors for the weak formulation.

```
1 def Assemble(Triangulation, Basis, x_q, w_q, equation_type,
2             Input, *argv):
3     w_q = w_q/(np.sqrt(27)/4) * 1/2
4
5     global_dof = Basis.GlobalDOF(Triangulation);
6     ConnectMatrix = Basis.GlobalMapping(Triangulation)
7
8     if equation_type == 'Kohn-Sham':
9         M, W, V_eff = KohnShamSolver(Input, Basis, x_q, w_q,
10                                     ConnectMatrix, global_dof, argv)
11         return (1/2 * W + V_eff), M
12     elif equation_type == 'Poisson':
13         P, f = PoissonSolver(Input*4*np.pi, Basis, x_q, w_q,
14                              ConnectMatrix, global_dof, argv)
15         return P, f
16     elif equation_type == 'Heat':
17         M, W, b = HeatSolver(Input, Basis, x_q, w_q,
18                              ConnectMatrix, global_dof, argv)
19         return M, W, b
```

Listing 3.7: The *Assemble* function which initialize all relevant assemblies for a given equation type.

The input variable given in line 1 differ depending on the type of equation. For the Kohn-Sham equations it is the external potential, for Poisson's equation it is the electronic density and finally as the source term in the heat equation. Inputs such as the basis module and quadrature coordinates and weights could easily be loaded within the module. For now it left as an input to give the user full control and allow for easily accessible comparison. The triangulation in used in the software is can be defined thorough an *\*argv* input. Simply by passing the string 'dual' as the last input if the system purely consits of equilateral triangles. Taking the example of the Kohn-Sham equations the function enters the if statement in line 8 which calls the function *KohnShamSolver* which reads

```

1 def KohnShamSolver(Input_pot , Basis , x_q, w_q, ConnectMatrix ,
2                     global_dof):
3     from scipy.sparse import lil_matrix;
4     local_dof = Basis.LocalDOF()
5     M = lil_matrix((global_dof , global_dof))           #Mass matrix
6     W = lil_matrix((global_dof , global_dof))           #Stiffness matrix
7     V = lil_matrix((global_dof , global_dof))           #Potential matrix

9     W_local = StiffnessLocal(Basis , x_q, w_q)
10    M_local = MassLocal(Basis , x_q, w_q)
11    V_local = PotLocal(Basis , x_q, w_q, Input_pot)

13    for K in range(ConnectMatrix.shape[0]):
14        glo_indices = Local2Global(ConnectMatrix , K)
15        for i in range(local_dof):
16            for j in range(local_dof):
17                W_global._set_intXint(glo_indices[i],glo_indices[j],W_local[i,j]+\
18                                     W[glo_indices[i], glo_indices[j]])
19                M_global._set_intXint(glo_indices[i],glo_indices[j],M_local[i,j]+\
20                                     M[glo_indices[i], glo_indices[j]])
21                val = 0
22                for q in range(len(w_q)):
23                    poly_expansion = continuous_rep(Input_pot , glo_indices , Basis ,
24                                                     x_qdinates[q,:])
25                    val += V_local[i,j + q*local_dof] * poly_expansion
26                    V._set_intXint(glo_indices[i], glo_indices[j], val +\
27                                   V[glo_indices[i], glo_indices[j]])
28    return M, W, V

```

**Listing 3.8:** The *KohnShamSolver* function assemble  $M$ ,  $W$  and  $V$  necessary for solving the Kohn-Sham equations.

Line 5-8 initializes the matrices that in the end will be returned. Due to the sparse nature of the matrices *scipy.sparse* is used to minimize the memory usage. Line 9-11 calls functions *StiffnessLocal*, *MassLocal* and *PotLocal* which return the local matrices. If the triangulation with added vertices is used two local matrices are computed from the *UnitCell* module.

Assembly of  $M$  and  $W$  then comes down to adding the contributions in the correct matrix entry. The entries of the array from line 14 are the global nodes defining the triangle  $K$ . Adding contributions to the sparse matrices is done in line 17-20, here *.\_set\_intXint* takes the matrix entries in the first two inputs and then the value. The value will then be local matrix contribution added to the current value of the global matrix entry.

The assembly of  $V$  is however a bit more complicated. For each updated value of  $V$  in line 26 a loop over the number of quadrature points is needed as seen in line 22. Taking an example of a specific triangle and combination of  $\varphi_i\varphi_j$  we find the polynomial expansion value for the effective potential in line 23. Iterating with  $q$  through all quadratures points and adding contributions can be done, by the entries  $V\_local[i,j + q*local\_dof]$  since  $\varphi_i\varphi_jw_q$  for each quadrature is present. Then the integral contribution for a specific node is found in line 25 and added to the sparse matrix in line 26. The functions for Poisson's equation and the heat equation are computed similarly.

### Cotangent Laplacian

The cotangent laplacian introduced in section 3.1.5 is constructed entirely through information about the triangulation. It is applicable when using a linear basis. It is computed by the function *Cotangent\_Laplacian* and in the case of the dual reads

```

1 def Cotangent_Laplacian(Basis, dual_faces):
2     global_dof = Basis.GlobalDOF(dual_faces)
3     Cotan_Laplacian = lil_matrix((global_dof, global_dof))
4     A_mat = lil_matrix((global_dof, global_dof))
5
6     #single cell Voronoi contribution
7     A = np.sqrt(27) / 4 / 3
8
9     w_ij = 1/np.sqrt(3)
10    for i in range(global_dof):
11        triangle_ind = np.where(dual_faces == i)[0]
12        ind = np.unique(dual_faces[triangle_ind])
13        for j in ind:
14            Cotan_Laplacian._set_intXint(j, i, -w_ij)
15
16    #Assigning values the diagonal element for both matrices
17    number_of_neighbours = len(np.where(dual_faces == i)[0])
18    Cotan_Laplacian._set_intXint(i, i, w_ij * number_of_neighbours)
19    A_mat._set_intXint(i, i, A * number_of_neighbours)
20
21    return Cotan_Laplacian, A_mat

```

Listing 3.9: Cotangent\_Laplacian function for computing the matrices relevant for the curved space stiffness matrix.

Here the single cell Voronoi contribution is defined in 7, which is  $1/3$  of the area of the equilateral triangle the dual representation consists of. An important thing to note is that a finer triangulation within the original dual representation requires a scaling of this area. The for loop initialized in line 10 will for each neighbouring triangular indices  $(i, j)$  insert  $-w_{ij}$  into the cotangent Laplace matrix in line 14. By computing the number of neighbours triangle  $i$  has in line 17, we can fill in the diagonal elements in both cotangent Laplacian and the area matrix in line 18 and 19. This implementation makes no restrictions when constructing the Laplacian matrix to the number of neighbours a triangle may have, which makes this directly applicable to fulleroid structures.

## 3.3 Constructing a 2-Dimensional DFT with FEM Software

Now that the FEM modules which allow for solving PDEs are described we turn the attention to towards developing a DFT on the manifolds. This chapter will start by presenting the approach to the exchange-correlation in which a local density approximation has been implemented. Section 3.3.3 will then explain the general approach of the self consistent field(SCF) loop within DFT. Section 3.3.4 will then show the SCF loop implemented in this work, where the PDE solver is implemented to solve Poisson's equation and the Kohn-Sham equations in an iterative manner.

### 3.3.1 Exchange-Correlation Potential

The degree of complexity within a DFT implementation heavily depend on the approximation used for the exchange-correlation energy functional  $E_{xc}$ . A few examples of different groups of approximations in order of increasing complexity are: local density approximations(LDA) which only depend on the the spatial value of the density, generalized gradient approximations(GGA) which take the gradient behaviour of the density into consideration and meta GGA which takes the behaviour of the Laplace operator on the Kohn-Sham orbitals into account. Other implementations exist and still much ongoing research for functionals that yield chemical accuracy.

Using the functional  $E_{xc}[\rho]$  the exchange-correlation potential  $v_{xc}$  can be found by taking the functional derivative of the two corresponding exchange and correlation energy terms i.e.

$$v_{xc}(\rho) = v_x(\rho) + v_c(\rho) = \frac{\delta E_x[\rho]}{\delta \rho} + \frac{\delta E_c[\rho]}{\delta \rho}. \quad (3.3.1)$$

We therefore now wish to incorporate some expressions for the functionals  $E_x[\rho]$  and  $E_c[\rho]$  which with the help of functional derivatives will yield the exchange-correlation potential.

### 3.3.2 Local Density Approximations

LDA<sup>3</sup> approaches are in general simple, but can often yield useful results. Many of the approaches are based of the the uniform electron gas model also known as jellium. Here an electronic density is treated in a confined space with a constant potential. This yield a constant electronic density which is solved with a Hamiltonian accounting for electron-electron interactions. The functional  $E_x[\rho]$  for jellium actually has an analytical expression[21] of

$$E_x[\rho] = \int \rho(\mathbf{x}) \epsilon_x(\rho) d\mathbf{x} = -\frac{3}{4} \left( \frac{3}{\pi} \right)^{1/3} \int \rho^{4/3}(\mathbf{x}) d\mathbf{x}, \quad (3.3.2)$$

where  $\epsilon_x(\rho)$  is the exchange energy per particle of a uniform electron density gas. Note atomic units are used throughout this work. Finding the potential expression follow from equation (3.3.1) by taking the functional derivative with respect the density. Using the general formula for functional derivatives in

---

<sup>3</sup>Accounting for spin we refer to it as local spin density approximation LSDA.

equation (2.3.25) is simple, since  $E_x[\rho]$  does not depend on the spatial derivative of  $\rho$ . It can therefore be found simply by taking the derivative with respect to  $\rho$ , like

$$v_x(\rho) = \frac{\delta E_x[\rho]}{\delta \rho} = \frac{\partial}{\partial \rho} \left( -\frac{3}{4} \left( \frac{3}{\pi} \right)^{1/3} \rho^{4/3}(\mathbf{x}) \right) = - \left( \frac{3}{\pi} \rho(\mathbf{x}) \right)^{1/3}, \quad (3.3.3)$$

thus yielding the potential accounting for the exchange.

For the correlation energy functional in LDA analytical expressions only exists in the limiting cases of high and low density systems. Luckily several numerical stochastic methods [22] has produced correlation energy results in between the limits. Based on this several fits have been suggested to encapsulate the limits as well as the simulated discrete points. Here the fit suggested by Chachiyo in [23] for  $\epsilon_c(\rho)$  is used. It has the form

$$\epsilon_c(\rho) = a \ln \left( 1 + \frac{b}{r_s} + \frac{b}{r_s^2} \right) \quad \text{with} \quad r_s = \left( \frac{3}{4\pi\rho(\mathbf{x})} \right)^{1/3}, \quad (3.3.4)$$

where the dimensionless  $r_s$  is known as the Wigner-Seitz density parameter with the high-density limit being  $r_s \rightarrow 0$  and low-density limit  $r_s \rightarrow \infty$ . In (3.3.4)  $a$  and  $b$  are constant defined to be  $a = \frac{\ln(2)-1}{2\pi^2}$  and  $b = 20.4562557$ . The correlation energy functional reads

$$E_c[\rho] = \int \rho(\mathbf{x}) \epsilon_c d\mathbf{x}, \quad (3.3.5)$$

here the functional derivative follows from the same argument as for the exchange functional thus

$$v_c(\rho) = \frac{\partial}{\partial \rho} (\rho(\mathbf{x}) \epsilon_c(\rho)) = \epsilon_c \frac{\partial}{\partial \rho} \rho(\mathbf{x}) + \rho(\mathbf{x}) \frac{\partial}{\partial \rho} \epsilon_c(\rho) \quad (3.3.6)$$

The first term is just equal to  $\epsilon_c(\rho)$ . Recall that  $\frac{d}{dx} \ln u = \frac{1}{u} \frac{du}{dx}$ , we then turn our attention to the second term above and introduce the constant  $c = \left( \frac{3}{4\pi} \right)^{1/3}$ , the term then reads

$$\rho(\mathbf{x}) \frac{\partial}{\partial \rho} \epsilon_c(\rho) = \frac{a\rho(\mathbf{x})}{1 + \frac{b\rho^{1/3}(\mathbf{x})}{c} + \frac{b\rho^{2/3}(\mathbf{x})}{c^2}} \frac{\partial}{\partial \rho} \left( 1 + \frac{b\rho^{1/3}(\mathbf{x})}{c} + \frac{b\rho^{2/3}(\mathbf{x})}{c^2} \right) \quad (3.3.7)$$

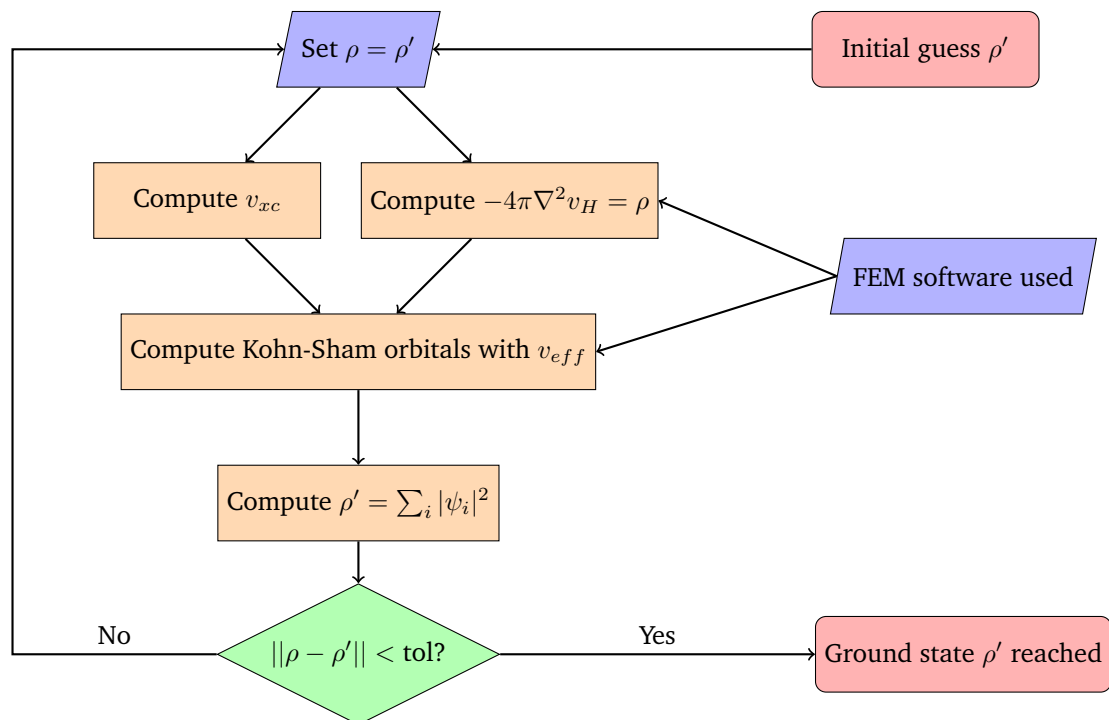
$$= \frac{a\rho(\mathbf{x})}{1 + \frac{b\rho^{1/3}(\mathbf{x})}{c} + \frac{b\rho^{2/3}(\mathbf{x})}{c^2}} \left( \frac{b}{3c\rho^{2/3}(\mathbf{x})} + \frac{2b}{3c^2\rho^{1/3}(\mathbf{x})} \right), \quad (3.3.8)$$

Yielding a correlation potential which in its full glory now reads

$$v_c = a \ln \left( 1 + \frac{b\rho^{1/3}}{c} + \frac{b\rho^{2/3}}{c^2} \right) + \frac{a\rho}{1 + \frac{b\rho^{1/3}}{c} + \frac{b\rho^{2/3}}{c^2}} \left( \frac{b}{3c\rho^{2/3}} + \frac{2b}{3c^2\rho^{1/3}} \right). \quad (3.3.9)$$

### 3.3.3 The Self-Consistent-Field Loop

Now with the FEM software and a theoretical framework for the exchange-correlation potential in place, applying it in a DFT routine is the task at hand. The hope is to create an algorithm that can find a ground state electronic density for a fullerene manifold. This can be achieved by solving the Kohn-Sham equations in an iterative manner. This is achieved in DFT by a method called the self-consistent field(SCF) loop. The computational scheme of this method is illustrated in figure 3.8 and will if successfully integrated yield a ground state density. In this case the effective Kohn Sham potential solved in the Kohn-Sham equations is  $v_{eff} = v_H + v_{xc}$ . Usually the positively charged nuclei environment would be included in  $v_{eff}$  and would be passed as a pre-determined input. This is obviously crucial to extract any useful ground state density from the fullerene, however this is beyond the scope of this thesis.



**Figure 3.8.:** The self consistent field loop takes in an initial density outputs a ground state density for the system. This is done by solving the Kohn-Sham equations with an effective potential  $v_{eff}$  computed using the density  $\rho$ , which computes a new density  $\rho'$  using the Kohn-Sham orbitals, if  $\rho$  and  $\rho'$  have not converged the process repeats using the new density.

### 3.3.4 Computing the SCF loop

The relevant inputs when implementing the scheme in figure 3.8 in a function will among other things be the initial density, number of electrons present and the tolerance parameter used as a convergence criteria. The script *DFT\_Self\_Consistent\_Loop.py* contain the function *DFT\_SCF* which with specified input solve the SCF-loop, it reads

```
1 def DFT_SCF(triangulation, tolerance, rho_initial, occupied_orb, Basis):
2     import FEM.FEM_Assembly as FEM_Assembly
3
4     #Normalizing the initial density
5     rho_const = full_integration(rho_initial, coordinates_54, weights_54,
6                                 triangulation, Basis)
7     rho_old = rho_initial/rho_const * occupied_orb
8
9     global_dof = Basis.GlobalDOF(triangulation);
10    counter = 0
11    while True:
12        #Computing the Hartree Potential
13        P, f = FEM_Assembly.Assemble(triangulation, Basis, coordinates_54,
14                                    weights_54, 'Poisson', -rho_old, 'dual')
15        v_h = linalg.lsqr(P,f)[0]
16        v_h -= np.abs(np.max(v_h))
17
18        #Assembling Kohn-Shal potential and solving the Kohn-Sham equations
19        v_eff = v_effective(v_h, rho_old)
20        A, B = FEM_Assembly.Assemble(triangulation, Basis, coordinates_54,
21                                    weights_54, 'Kohn-Sham', v_eff, 'dual')
22        energy_eigen, orbitals_eigen = linalg.eigs(A, k=200, M=B, which='SM')
23
24        #Picking the eigenvectors/orbitals with the smallest energy/eigenvalue
25        index = np.argsort(energy_eigen)
26        orbitals_eigen_small = orbitals_eigen[:, index[0:occupied_orb]]
27
28        #Normalizing orbital densities
29        orbitals_squared = (orbitals_eigen_small * \
30                            np.conj(orbitals_eigen_small)).real
31        for k in range(occupied_orb):
32            rho_norm = full_integration(orbitals_squared[:,k], coordinates_54,
33                                      weights_54, triangulation, Basis)
34            orbitals_squared[:,k] /= rho_norm
35
36        #Computing the full density
37        rho_new = np.sum(orbitals_squared, axis=1)
38        rho_new = alpha*rho_new + (1-alpha)*rho_old
39
40        #Checking if convergence tolerance is met
41        if np.linalg.norm(rho_new - rho_old, 2) < tolerance:
42            print('Convergence at ' + str(counter) + ' iterations')
43            return rho_new
```



**Listing 3.10:** The function *DFT\_SCF* which calculates a ground state density using the SCF-loop method. The loop iterates will run until the given tolerance criteria is met.

The input *triangulation*<sup>4</sup> in the above listing refer to the cubic graph sparse adjacency matrix and *occupied\_orb* is the number of electrons present occupying the Kohn-Sham orbitals. Before the SCF loop starts the input density is made sure to integrate to the number of occupied orbitals in line 5-7. Here the function *full\_integration* calculates the integral over the entire domain of a given function using quadrature points and the polynomial expansion of the function.

Entering the while loop in 11, where the loop start by computing relevant matrices and vectors for calculating Poisson's equation in line 13. A least square solver<sup>5</sup> is used in line 15 for computing the Hartree potential. This yield a potential with values around zero containing both positive and negative values. This will create the same physical behaviour within the system if only accounting for Coulomb electron-electron repulsion, but will be problematic when accounting for other potential terms in the Kohn-Sham potential. The next operation is therefore to shift the potential to have a maximum value of 0 in line 16.

The Hartree potential is the fed to the function *v\_effective* in line 19. Here the effective potential is computed using the Hartree potential and constructing the exchange-correlation potential as in equations (3.3.3) and (3.3.9) using the density *r\_old*. The potential *v\_eff* is used in line 19 as an input into the FEM assembly for the Kohn-Sham system. This leaves the Kohn-Sham eigenvalue problem to be solved. Using the sparse eigenvalue solver *linalg.eigs* and requiring it, to return a sufficiently large number of solutions. The outputs are then a vector and a matrix with the eigenvalues(eigen-energies) and their corresponding eigenvectors(orbitals) respectively. The eigenvectors of interest are those of the lowest energy up to the number of orbitals included in the system. All orbital densities are then computed and normalized in line 29-34 by forcing the integration of each orbital density to be 1 over the entire domain.

A new density based on the occupied orbitals is then computed in line 37. The density is then tweaked using density mixing in line 38, which will be explained shortly. If the norm of the differences for the new and old density is within the tolerance the new density is returned, otherwise it iterates through the while loop again using the new density<sup>6</sup>.

A general problem within the SCF loop is the occurrence of charge sloshing. This is a problem in which charges bounce back and forth in the system mainly due to the Hartree potential. This can occur in various DFT settings and this implementation is especially susceptible to this, since no positive charges are present to create attraction. A way to counteract this behaviour is through density mixing. The somewhat self explanatory name will tweak the newly calculated density in each iteration by mixing it with the density from the former iteration. A simple way to implement this is through a factor  $\alpha$  between 0 and 1 with the density computed as

$$\rho'_{i+1} = \alpha\rho_{i+1} + (1 - \alpha)\rho_i, \quad (3.3.10)$$

yielding  $\rho'_{i+1}$  used to further iterate. This can in theory speed up the DFT calculations greatly, especially if the SCF loop is close to an energetic minima but overshoot this minima in each iteration. This scheme

<sup>4</sup>This example uses the dual representation which is passed into the FEM functions in line 12 and 21.

<sup>5</sup>the sparse least square solver *linalg.lsqr* from the *scipy.sparse* library to be exact

<sup>6</sup>This snippet is slightly modified for compactness sake. Code loading quadrature data, importing basis module and abort statements for the while loop is excluded.

was used with  $\alpha = 0.5$  and section 4.2.3 will show results where the behaviour with and without density mixing will be highlighted during simulations.

Energy functionals evaluating the energy within the manifold will be a sum of contribution arising from kinetic and potential terms i.e.

$$E[\rho] = E_T[\rho] + E_H[\rho] + E_X[\rho] + E_C[\rho]. \quad (3.3.11)$$

This will be presented for various solutions of the SCF loop. Here the kinetic energy will be calculated using the Thomas-Fermi[21] orbital kinetic functional of

$$E_T[\rho] = \frac{3}{10} (6\pi^2)^{2/3} \int_{\Omega} \rho^{5/3}(\mathbf{x}) .d\mathbf{x}. \quad (3.3.12)$$

### 3.4 Solving the Heat Equation

To investigate the validity of the FEM implementation we wish to simulate the heat equation on manifold surfaces. The heat equations describes the diffusive evolution of a spatial and time dependent function  $u(\mathbf{x}, t)$ , and in our particular 2-dimensional case reads

$$\frac{\partial}{\partial t}u(\mathbf{x}, y, t) = \alpha \nabla^2 u(\mathbf{x}, y, t), \quad (3.4.1)$$

where  $\alpha$  is a diffusivity constant of the system. The time evolution for the simulation is handled by a simple forward Euler method. This approach seemed fitting since it is easily implemented and no general benefit would come of a more complex method, since the goal of the simulations were to verify and present the FEM software as a general solver. The evolution is performed by obtaining  $u(\mathbf{x}, y, t_{n+1})$  based on function information  $u(\mathbf{x}, y, t_n)$  by

$$\frac{u(\mathbf{x}, y, t_{n+1}) - u(\mathbf{x}, y, t_n)}{\delta} = \alpha \nabla^2 u(\mathbf{x}, y, t_n) \Rightarrow \quad (3.4.2)$$

$$u(\mathbf{x}, y, t_{n+1}) = u(\mathbf{x}, y, t_n) + \delta \alpha \nabla^2 u(\mathbf{x}, y, t_n), \quad (3.4.3)$$

where  $\delta$  denote the time-step.

To simplify the expression we introduce the notation for the the heat distribution  $u(\mathbf{x}, y, t_{n+1}) \Rightarrow u_n$ . To state the problem in the weak formulation we once again start by taking the integral over each term multiplied by a test function

$$\int_{\Omega} u_{n+1} v d\mathbf{x} = \int_{\Omega} u_n v d\mathbf{x} + \delta \alpha \int_{\Omega} \nabla^2 u_n v d\mathbf{x}. \quad (3.4.4)$$

Following the arguments from section 2.2.2 we end up with a vectorized problem of the form

$$\mathbf{M} \mathbf{u}_{n+1} = \mathbf{M} \mathbf{u}_n - \delta \alpha \mathbf{L} \mathbf{u}_n, \quad (3.4.5)$$

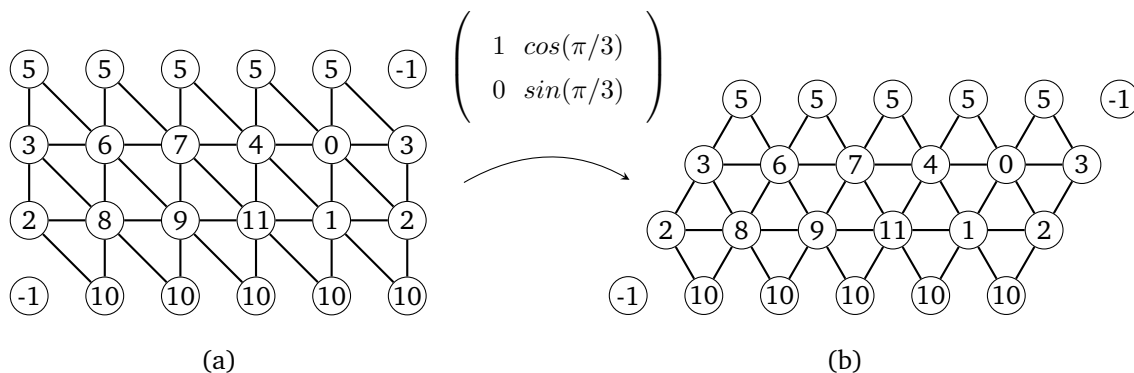
with the matrices

$$\mathbf{M} = \int_{\Omega} \varphi_i \varphi_j d\mathbf{x} \quad \text{and} \quad (3.4.6)$$

$$\mathbf{L} = \int_{\Omega} \nabla \varphi_i \cdot \nabla \varphi_j d\mathbf{x} \quad (3.4.7)$$

Thus  $\mathbf{u}_{n+1}$  is obtained by solving a linear system of equations. The FEM matrices needed are independent of the function  $u(\mathbf{x}, y, t_n)$  and only depend on the geometry. The matrices can thus be obtained in the initialization of the simulation and used throughout. The simulation is initialized by constructing some initial distribution  $\mathbf{u}_0$  and then let the system propagate in time by updating  $\mathbf{u}_n$  in each iteration. The simulation is then simply run until the system has converged to an even distribution. Evaluating the heat equation was performed to validate the FEM implementation. It has a quite intuitive behaviour and a visualization of the implementation was used to verify the geometrical interactions of neighbouring triangles and nodes. Animations<sup>7</sup> are produced in matplotlib to visualize the simulations performed.

<sup>7</sup>Gifs to be exact, short for for graphic interchange format.



**Figure 3.9.:** (a) A possible grid for a  $C_{20-I_h}$  unfolding, by using the rotational matrix yield (b) which is the unfolding on the Eisenstein plane consisting of equilateral triangles.

## 3.5 Visualization

As explained in section 2.1.3 the dual triangulation can be unfolded onto a 2-dimensional plane using Eisenstein integers, which will be used to visualize the results for solutions to differential equations. An example of such an unfolding with labelled vertices can be seen in figure 3.10 for the case of  $C_{20-I_h}$ .

### 3.5.1 Dual Unfolding

Among the geometrical fullerene data given are information of a possible unfolding. The necessary data to perform this unfolding onto a 2-dimensional coordinate system are the following: the dual triangulation, a labelled coordinate grid explained shortly, an array arcs containing all entries with directed edges and the corresponding arcpes which contains the geometrical coordinates in the grid defining a directed edge. It is important to note that this data is necessary for visualizations of the solutions, which are computed only using the graph information.

The grid given is a matrix where each entry is a labelled vertex, which for the  $C_{20-I_h}$  can take the form

$$\begin{pmatrix} 5 & 5 & 5 & 5 & 5 & -1 \\ 3 & 6 & 7 & 4 & 0 & 3 \\ 2 & 8 & 9 & 11 & 1 & 2 \\ -1 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}, \quad (3.5.1)$$

where  $-1$  denotes a matrix entry without a vertex. The cubic graph connectivity this this matrix represents is shown in figure 3.9

The coordinates for the unfoldings in the Eisenstein plane can be computed by applying the rotational matrix seen in figure 3.9. The Eisenstein coordinates for the unfolding are produced by calling the function *transformedEisenstein*, which reads

```

1 def transformedEisenstein(dual_faces, arcs, arcpes):
2     data = []; data_trans = []
3     import numpy as np
4     for tri in dual_faces:
5         tri_data = []; rot_tri = [];
6         j = 0; tri_array = np.zeros([3,2]); b = 0
7         while True:
8             if tri[0] == arcs[j][0] and tri[1] == arcs[j][1]:
9                 tri_array[0,:] = rot_func(arcpes[j][0])

```

```

10     tri_array[1,:] = rot_func(arcpos[j][1])
11     k = 0
12     while True:
13         if tri[1] == arcs[k][0] and tri[2] == arcs[k][1]:
14             tri_array[2,:] = rot_func(arcpos[k][1])
15             b = 1
16             break
17         k+=1
18     j+=1
19     if b == 1:
20         break
21     #data.append(tri_data); #data_trans.append(rot_tri)
22     data_trans.append(tri_array)
23     return data_trans

```

**Listing 3.11:** Function calculating Eisenstein coordinates for an unfolding, provided dual faces, directed edges and their coordinates.

By looping through all the triangles in the dual starting in line 4 the transformed coordinates for each triangle are calculated. The dual array obey the nature of directed edges, as an example in figure 3.9b an edge between label 2 and 3 appear twice, but in the dual array one triangle will have a directed edge going from 2 to 3 and another going from 3 to 2.

Searching for the triangle defined by [0,3,2] as an example, the if statement in line 8 finds the index  $j$  for a directed edge going from 0 to 3. This index is passed to the coordinates of the edge in `arcpos`, and the Eisenstein coordinates are computed using the rotational matrix from figure 3.9. The computation is repeated to find the coordinates of the last vertex in the triangle. With the unfolding in place the question is how to create a pixel map showing the results on the unfolding.

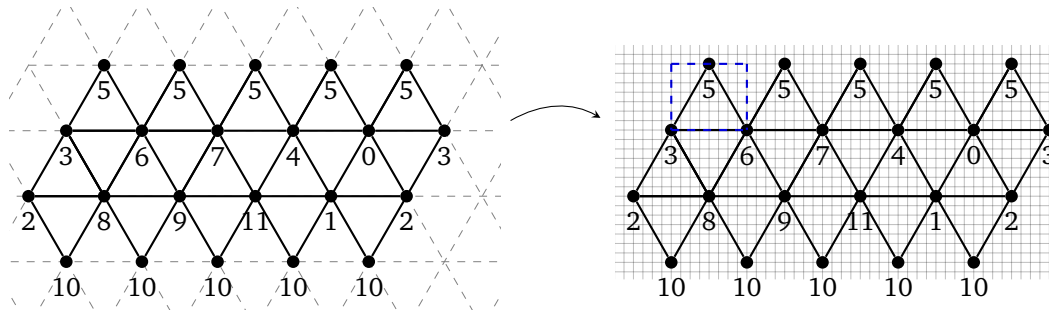
### 3.5.2 2-Dimensional Visualization

Creating a pixel map and assigning function values will be done using barycentric coordinates, for which a short explanation follow. Imagine a triangle with corners  $x_1$ ,  $x_2$  and  $x_3$  and denoting the area of this  $A$ . Placing a point  $x$  within the triangle one can construct 3 new triangles. Each corner will be associated with a barycentric coordinate  $\lambda_i$ , which is the ratio of the newly constructed triangle area opposite of corner  $x_i$  to  $A$ <sup>8</sup>. The barycentric coordinates can be calculated for all possible points  $x$ . For a point within the triangle the three barycentric scalars obey  $0 \leq \lambda_i \leq 1$  and  $\lambda_1 + \lambda_2 + \lambda_3 = 1$ . An easily computed transformation from Cartesian coordinates  $(x, y)$  to barycentric coordinates is

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \boldsymbol{\lambda} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (3.5.2)$$

The first step in the visualization is to create a pixel grid for the unfolding as seen applied in figure 3.10b. We now take the example of applying pixel values to a specific triangle of [5,6,3]. The coordinates defining the triangle are known and are used to create a bounding box of all pixels in the vicinity of the triangle, with an example shown in figure 3.10b. Looping through all pixels and calculating the barycentric coordinates for the center of the pixel, will show whether or not the pixel center is within the triangle. If not, then the loop continues without adding a pixel value. If all barycentric coordinates

<sup>8</sup>When it comes to giving an understanding of barycentric coordinates this is only one of many. This short explanation was picked due to it's quite intuitive nature.

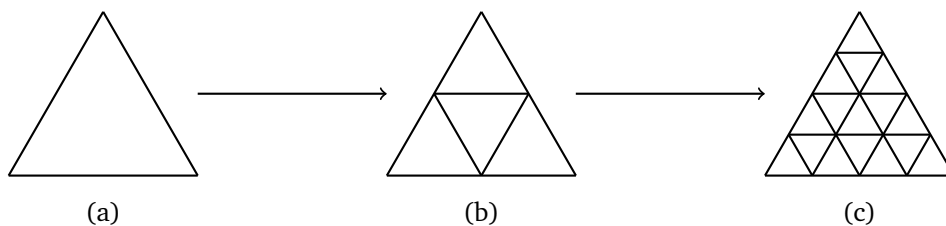


**Figure 3.10.:** (a) A 2-dimensional unfolding of the small  $C_{20-I_h}$  fullerene onto the Eisenstein plane consisting of equilateral triangles. (b) Adding a pixel grid to the unfolding, where each pixel value inside the unfolding is computed based on the triangle in which it is present. The blue dashed line represent a bounding box for the triangle [5,6,3]

lie between 0 and 1 for a pixel in a specific triangle a pixel value is added. This is done by transforming to Cartesian coordinates through equation (3.5.2), and using the function *continuous\_rep* introduced in 3.2.3. This calculates the polynomial representation of  $(x, y)$  using the triangle node values<sup>9</sup> thus yielding a pixel value **Appendix**. The full visualization is then achieved by repeating this for all triangles. The results presented in this thesis will all be in the dual representation due to this neat 2-dimensional unfolding allowing for visualization of solutions.

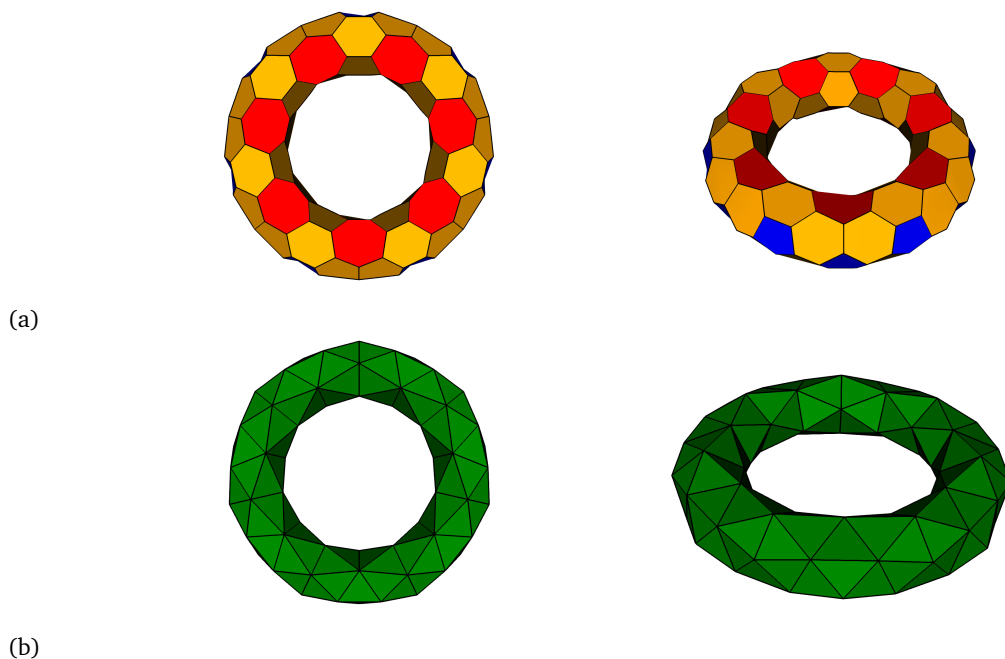
### 3.5.3 Mesh Refinement

The dual representation can easily be used to create a denser mesh. An equilateral triangle can be divided into 4 equilateral triangles all with half the side length of the original, this is shown in figure 3.11 going from (a) to (b). Computing the new sparse adjacency matrix is done using a the *GlobalMapping(T)* from the quadratic basis function implementation, module *BasisFunctions\_Quadratic* introduced in section 3.2.1. Here 3 nodes were added to represent all six shape functions on each triangular element. These labels can just be extracted and connected to create the adjacency for the triangulation. While this is neat it would be useless in this setting if the new solution could not be visualized. The function *Mesh\_quadruple* therefore take the dual faces, directed edges and their location as an input, and return the same for the new mesh. All new vertices are placed in between the relevant neighbours in the Eisenstein plane, leaving an identical unfolding. This function can be called  $n$  times yielding a mesh of  $N_f \cdot 4^n$  elements, illustrated in figure 3.11 going from 1 element (a), to 4 (b) and 16 (c). This routine can be called in the *FEM\_Assembly* module allowing for easily accessible mesh refinement.



**Figure 3.11.:** The computational result of the functions *Mesh\_quadruple*. Feeding a dual triangulation, directed edges of unfoldings and their coordinates on the Eisenstein grid will lead to each element being divided into four new elements. The function will for an original element (a) divide and create 4 elements (b) which run again yield 16 elements (c).

<sup>9</sup>Not necessarily just the vertices, but all nodes based on the polynomial order of the shape functions.



**Figure 3.12.:** (a) show the  $C_{168}$  torus shaped fulleroid consisting of 14 pentagons and 14 heptagons with chemically stable coordinates, where blue, orange and red illustrate the pentagon, hexagon and heptagon faces respectively. (b) is the corresponding dual representation computed from the coordinates of each face in (a)

### 3.6 Constructing a Torus Shaped Fulleroid

This work will also extend beyond fullerene manifolds by investigating a fulleroid structure. The FEM software is written generally enough to solve equations on fulleroid dual manifolds, since they are within cubic graph theory. The alternative triangulation would need a bit of generalization to extent to heptagons, so the focus will be on the dual representation.

A toroidal shaped fulleroid consisting of pentagons, hexagons and heptagons was chosen. This has a genus of ( $g = 1$ ) which will for the discrete Gauss-Bonet theorem yield  $\sum_{i=1}^n K_i = 4\pi(1 - g) = 0$ . Therefore the torus must have an equal number of pentagons and heptagons to form a closed structure. But to perform FEM calculations a dual array representing the torus is needed. Computation of the connectivity of the a torus was done using coordinates from *Hypothetical toroidal, cylindrical, and helical analogs of  $C_{60}$*  [13]. Here chemically stable coordinates are calculated for  $C_{120}$ ,  $C_{144}$  and  $C_{168}$ . The article states lists of seven 3-dimensional coordinates that through different given rotational operations yield all atomic coordinates. This work implemented the  $C_{168}$  shown in figure 3.12a.

Once the atomic structure was computed through the given rotations the indices defining a polygon was computed. This was achieved by starting from an atom and finding and picking a closest neighbours. Now that the two first indices are decided a direction can be enforced, meaning that the next the nearest neighbour picked has a direction right of the original direction<sup>10</sup>.

With the faces shown in figure 3.12a computing the dual faces is straightforward, since the dual coordinates are just the mean value of all 3-dimensional coordinates defining a specific polygon, and the dual faces are computed from the adjacency of the polygons. The dual faces array arising from figure 3.12b will be the input to the FEM software.

<sup>10</sup>This is perhaps an oversimplification, the code is found in the folder /Torus/ on GitHub along the rotational scheme for generating the coordinates, if of interest.

## Results and Discussion

The goal of the research project this thesis is within is, to be able to one day perform fast purely 2-dimensional DFT on fullerene surfaces. The speculation is that these 2-dimensional densities sufficiently reflect the nature of its true 3-dimensional counterpart. This new way of approaching quantum chemical problems, that the fullerene structures allow for, is hoped to eventually lead to understanding and fast computations of properties of specific fullerene isomers derived through the 2-dimensional electronic density. The aspiration of this coming together would permit for a multitude of fast computations that is simply not possible currently.

As a step towards this goal I have built a 2-dimensional FEM software library for discrete surface manifolds, which allows us to solve differential equations on the surfaces using only the fullerenes graph representation. The library is written in a generalized way that allow computations of PDEs on any cubic graph with periodic boundary conditions consisting of equilateral triangles, e.g. fulleroids. This section will start by presenting results produced using the FEM software, since it is a vital part of further work on the DFT implementation and has been a focus for the work at hand.

The first results presented are simulations of the heat equation. To help validate the implementation as well as aid in the understanding of the different basis function implementations. Here simulations with various initial distributions will be presented on the surfaces of  $C_{20}-I_h$  and a  $C_{60}$  nanotube. It is important to note that, while the surfaces the results will be presented for are of rather small fullerenes. Thus the assumption that the surface behaviour clearly dominates over the 3-dimensional interactions in the fullerene volume will generally be least applicable here. This will be crucial when investigating densities going forward, but is of no concern to this work.

With the validation from the heat equation in place the attention is turned to results related to the simple DFT described in section 3.3. Here Poisson's equation and the Kohn-Sham equations have to be solved in every iteration in the SCF loop. Understanding solutions to these equations before an iterative implementation is therefore vital. Electron potential solutions to Poisson's equations will be shown along the density from which it arose. Here the mesh dependence of the solutions is showcased as well. These will be solutions of the  $C_{20}-I_h$  surface using various basis implementations.

Orbitals arising from the Kohn-Sham equations are then presented. The presented orbitals of the non-interacting electrons have been computed in a setting of a constant potential. This allow for us to compare the orbitals to the well known solutions of the hydrogen atom by seeing the  $C_{20}-I_h$  surface as a crude triangulation of a sphere.

With the above solutions in place it will let us implement the FEM software in an iterative manner within the SCF loop. This will be the first step towards an iterative DFT calculation on the fullerene surfaces. Simulations of this will be presented with various electron numbers i.e. the number of orbitals of which the density arise. The simulations will be on the surfaces arising from the  $C_{20}-I_h$ ,  $C_{60}$  nanotube and a  $C_{120}-D_6$  surfaces are presented.



Note all results presented are manifolds arising from the dual representation unfolded onto the Eisenstein plane. This is due to the fact that the visualization for the dual is easily constructed in 2-dimensions, which is not the case for the alternative triangulation. The equilateral triangular nature of the dual also allow for simple mesh refinement within the same unfolding. Spatial units are defined in terms of  $A$  which is the area of the equilateral triangles the dual consists of.

## 4.1 Simulating the Heat Equation

The benefit of the simulations of the heat equations are that we can easily construct systems in which the final solution is known. From an initial heat distribution with no source and periodic boundary the solution needs to converge towards a uniform distribution. While the final solutions is important, it was mainly the flow of heat that aided in validating and debugging the assembled matrices. Dealing with the complexity of inter connecting nodes in a FEM software for high-order basis function implementations, is no easy task. Applying the matrices to simulations therefore aided greatly in validating and debugging the geometrical connectivity in the sparse matrices and vectors computed.

All simulations of the heat equation<sup>1</sup> are computed using forward Euler time integration, a simple first-order procedure. This will suffice, since the task at hand is not to produce numerical precise results but results of physical intuition. The snapshots presented from the computed simulations are snippets from movies<sup>2</sup> created which can be found at <https://github.com/SKS94/Fullerene-Thesis/tree/Gifs>, along with other simulations performed on various fullerenes.

### 4.1.1 Simulations on the $C_{20}$ - $I_h$ surface

#### Localized initial distribution

Starting by investigating similar simulations using the different basis function implementations shown in figure 4.1. The forward Euler simulations where performed with a similar localized initial heat distribution all denoted iteration 0 in the figure. Here snapshots of the distribution during the simulations are shown on the by now familiar  $C_{20}$ - $I_h$  unfolding. Figure 4.1 a, b and c are the flat linear, quadratic and curved linear implementations respectively. As expected from physical intuition all of these simulations eventually converge to a homogeneous distribution. The *full\_integration* function was called during the iterations to make sure the amount of heat in the system stayed constant<sup>3</sup>, which was the case for all the numerically stable heat simulations performed. After 20 iterations the flat space linear and quadratic simulations have dissipated to a somewhat alike distribution. The linear curved space simulation dissipate throughout the system slower which is highlighted at iteration number 20 and 70. The fact that the curved space simulation converges slower is illustrated in figure 4.2. The nature of the cotan Laplacian, which seem differently scaled to the other laplace operators, will be covered in section 4.2.1 when discussing the solutions to the Hartree potential.

In figure 4.2 the linear and quadratic flat space simulations are almost indistinguishable, while the curved space simulation converges significantly slower. Simulations can obviously be performed where the difference of the flat space implementations are apparent, but here the focus has been to show that a sufficiently small  $\delta$  yields somewhat identical results.

While discussing these continuous solutions in figure 4.1 remember that it is computed discretely using the number of nodes relevant to the basis functions. This discrete system only contain 20 cells which

<sup>1</sup>Perhaps referring to this as a solution to the diffusion equation would be slightly more correct, since this is a entirely dimensionless simulation performed.

<sup>2</sup>gifs

<sup>3</sup>i.e. conservation of energy

have  $N_{dof} = 12$  for (a), (c) and  $N_{dof} = 42$  for (b). This somewhat coarse mesh and the fact that the basis functions are piece-wise polynomial is quite apparent in the initial distribution (iteration 0). This can be seen by lines outlining edges in triangles connected to the "warmest" point.

### Random initial distribution

A simulation with a somewhat disordered initial distribution is shown in figure 4.3. The initial heat distribution is a collection of random numbers between 0 and 100 added to all nodes. Contrary to the system simulated in figure 4.1 which has a heat flow away from a localized source, this simulation has a more chaotic initial distribution. As the simulation iterates we once again arrive at an even distribution, while preserving the total amount of heat within the system. We see that the initial heat signature in figure 4.3a has mainly two areas that are significantly colder than the rest of the manifold. As the iterations progress in figure 4.3b and 4.3c we see that these spots remain colder until convergence is reached to an expected value of  $\approx 50 \text{ A}^{-1}$ .

### Heat simulation using various basis function implementations



**Figure 4.1.:** Snapshots of unit-less heat equation simulations performed on the surfaces arising from the  $C_{20}-I_h$  molecule. The simulations are performed using the written FEM software with various basis function implementations. All simulations uses a time-step  $\delta = 0.015$  and are initiated with the same initial heat distribution. The different basis function implementations shown are (a) the flat space linear, (b) the quadratic and (c) the linear curved space using the contangent Laplacian.

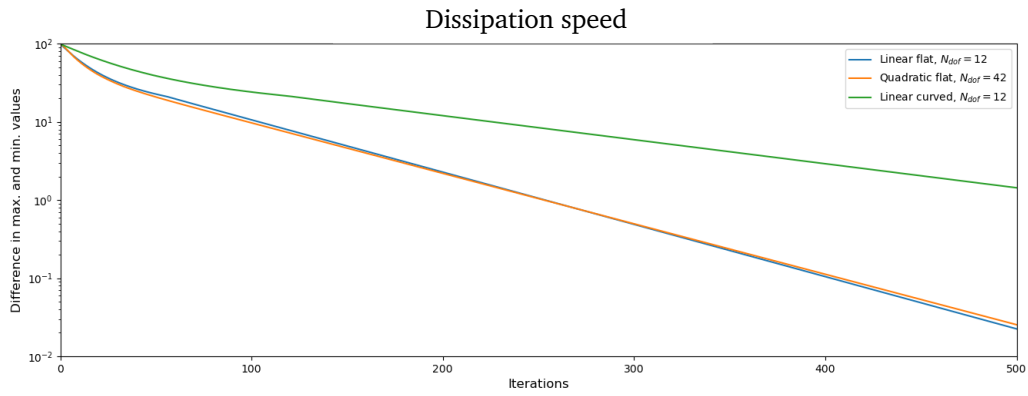


Figure 4.2.: The largest heat difference between all nodal points as a function of number of iterations in the heat simulations shown in figure 4.1.

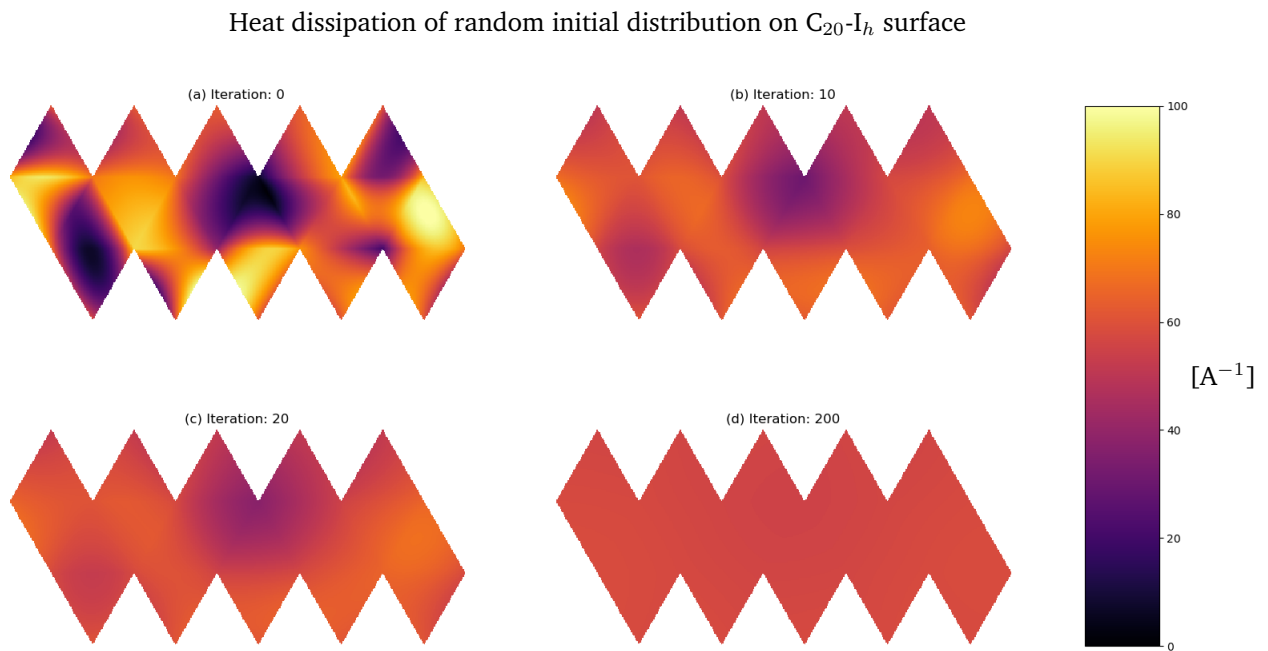


Figure 4.3.: A forward Euler simulation of the heat equation on manifold arising from the dual representation of the  $C_{20}\text{-}I_h$  with time-step  $\delta t = 0.015$ . Produced using the written FEM software with the quadratic shape function module. (a) The initial heat signature with random node values between 0 and 100 which after 10, 20 and 200 iterations yield (b), (c) and (d) converging to a uniform distribution.

## 4.1.2 Heat equation on a $C_{60}$ -Nanotube Surface

### Longest geodesic location using localized initial distribution

A similar simulation with an initial localized distribution is presented for the  $C_{60}$  nanotube surface. Which is the thinnest closed nanotube possible. All pentagons are situated at the endings, with 6 pentagons adjacent to each other. Since this is the first encounter with the  $C_{60}$  nanotube unfolding an illustration with labelled vertices appearing more than once is given in figure 4.4.

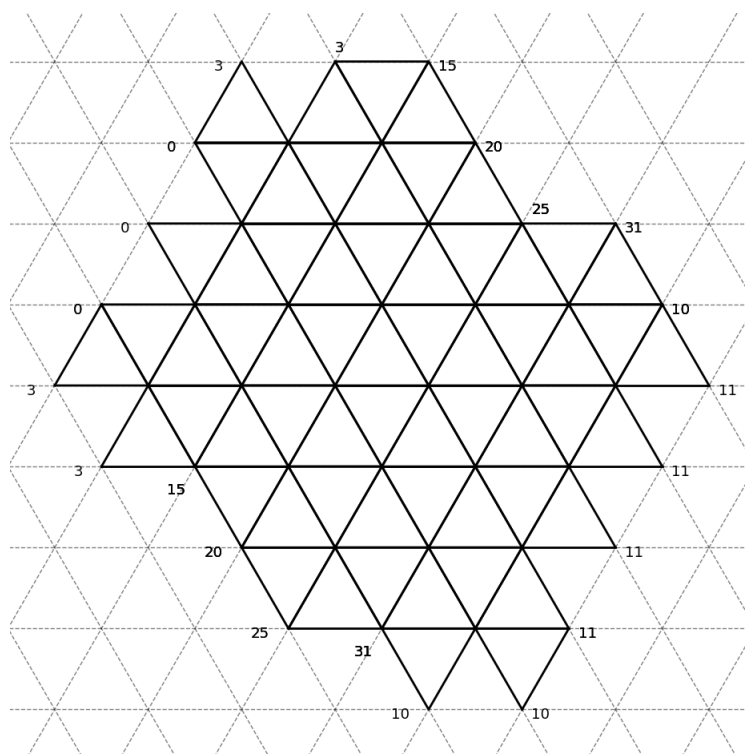
The specific simulation performed is like the localized distribution simulation performed for  $C_{20}$ - $I_h$ . The initial distribution can be seen in figure 4.5a. The purpose of this simulation will however be to showcase a different physical aspect of the system. A challenge when dealing with surfaces of a non-Euclidean geometry is calculate of basic geometrical properties, such as distances between two points. Approaching the challenge of finding the node with the longest geodesic to another node, where an initial heat distribution is placed. The heat will flow towards a uniform distribution, however the point furthest away will the last point the heat flow reaches.

A few snapshots of the simulation are presented in figure 4.6. The localized heat distribution is quite apparent after 40 and 60 iterations. After 100 iterations the heat is more scattered and the flow of heat perhaps not as intuitive to grasp. Our interest is mainly towards the near convergence plot in iteration 600. This is plotted with an appropriately scaled colormap in figure 4.5b. Here heat is still flowing towards the vertices of label 3 and 0. The unfolding reveal that the heat flow is closer to vertex number 3. Vertex 0 is therefore the longest geodesic vertex from the initial structure, which agrees with the unfolding.

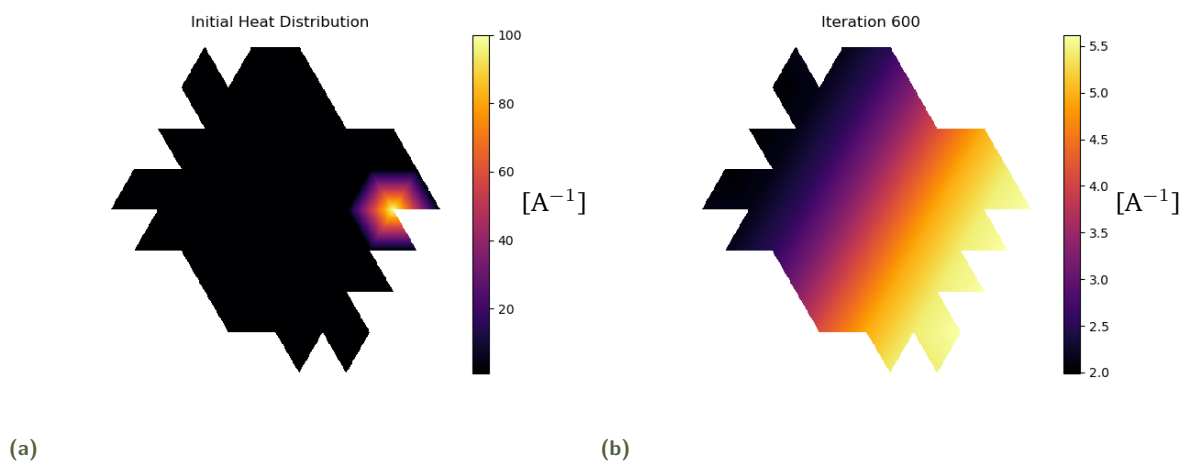
This may seem like a rather odd way to approach the stated challenge, but as a matter of the fact an approach subject to research within the computer science community<sup>4</sup>[24]. Now that the heat flow simulation using the FEM software has yielded results in accordance with physical intuition we turn our attention to the PDEs within the SCF-loop.

---

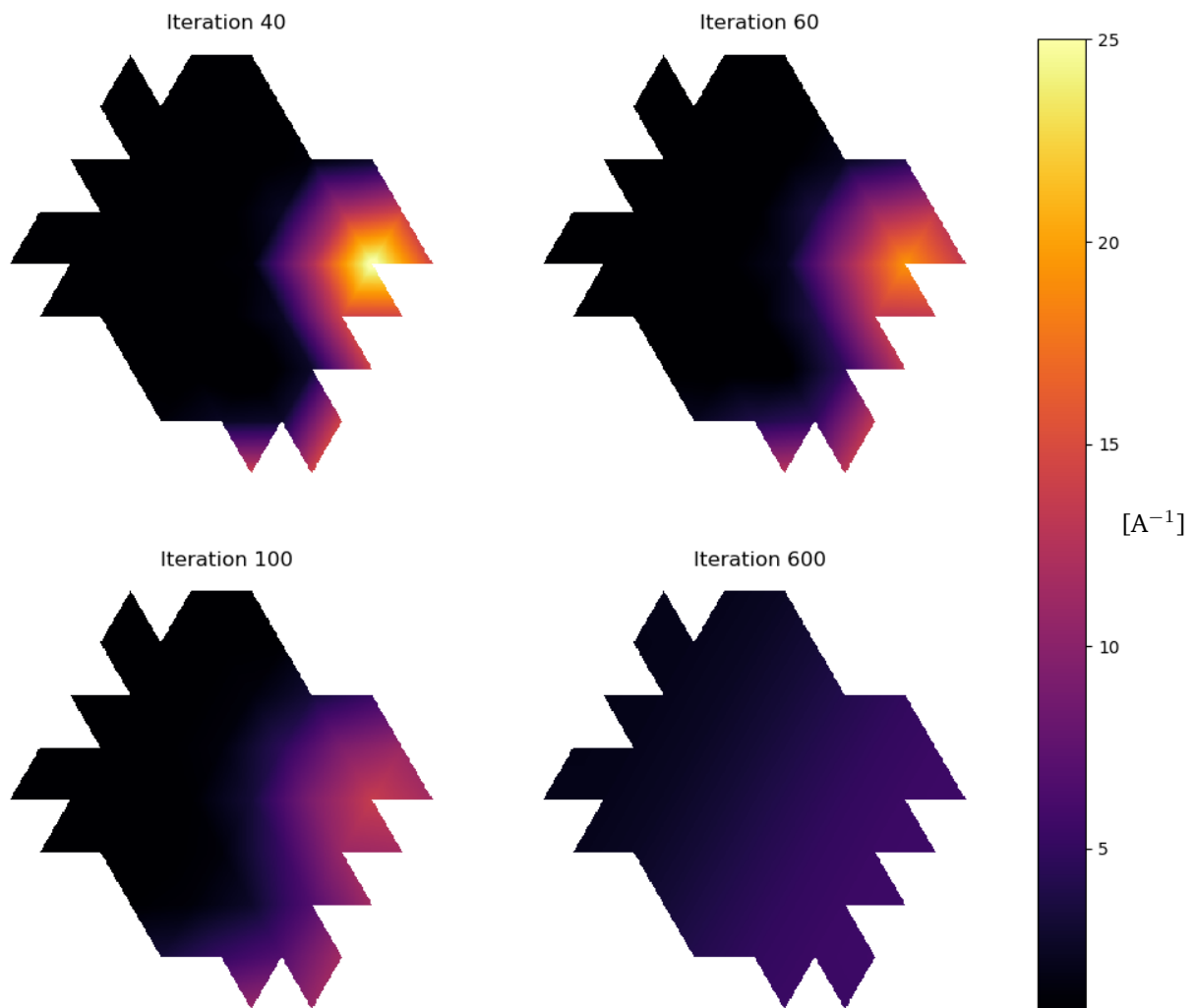
<sup>4</sup>As a way to help maps and compute behaviour in animations of objects with non-euclidean surfaces



**Figure 4.4.:** An unfolding of the the  $C_{60}$  nanotube, where all vertices that appear more than once in the unfolding are labelled. Connecting these matching labels will yield the 3-dimensional dual structure of the nanotube.



**Figure 4.5.:** (a) The initial heat distribution used to initialize the simulation shown in figure 4.6. (b) The simulation in figure 4.6 after 600 iteration shown with a colormap that highlights the differences in the heat distribution, which reveals the longest geodesic from the initial distribution



**Figure 4.6.:** Forward Euler simulation on the C<sub>60</sub> nanotube with  $\delta = 0.015$ . The snapshots shown are after 40, 60, 100 and 600 iterations were performed. For a detailed look at the distribution after 600 iteration consult 4.5b.

## 4.2 A Rudimentary 2-Dimensional DFT

Shifting the focus to the task of using the FEM software within a DFT model. This will aim to lay the groundwork of creating future physically sound simulations of the fullerene's electronic densities. Before diving into a SCF loop implementation in DFT, the following section will present solutions to Poisson's equation and the Kohn-Sham equations. The solutions to both Poisson's equation and the Kohn-Sham equations will be in a generalized setting i.e. solving systems in which we have at least some intuition of what to expect which allow to further validate the FEM software. Note that all units are with respect to the area  $A$  of the equilateral triangular unit cell for the dual. The energies are in Hartrees which will generally be denoted  $E_h$ .

### 4.2.1 Solutions for the Hartree potential

The DFT solutions produced by the SCF loop heavily depend on the implemented Kohn-Sham potential. One of the potential contributions is the Hartree potential, which can be computed by solving Poisson's equation for the electronic density. The Hartree potential is as discussed in section 2.3 a vital part in any DFT as it accounts for the repulsive nature of electron-electron interaction.

All solutions to the Hartree potential were found using a least square solver, as seen in the SCF-loop listing 3.10. The presented solutions of the Hartree potential will be illustrated with the corresponding density distribution from which it arose.

#### Local Density on $C_{20}$ - $I_h$ Surface

The first solution investigated is a localized density structure shown in figure 4.7. The density structures in figure 4.7 are all normalized to  $\int_{\Omega} \rho(\mathbf{x}) d\mathbf{x} = 10$ . The mid-edge nodes in the quadratic case have been set to match the values in the linear approach for the initial density. Once a solution in 4.7 was found it was shifted by subtracting the maximum value. The solution is therefore solely negative with the highest repulsion for negative charges being located at the high density<sup>5</sup>.

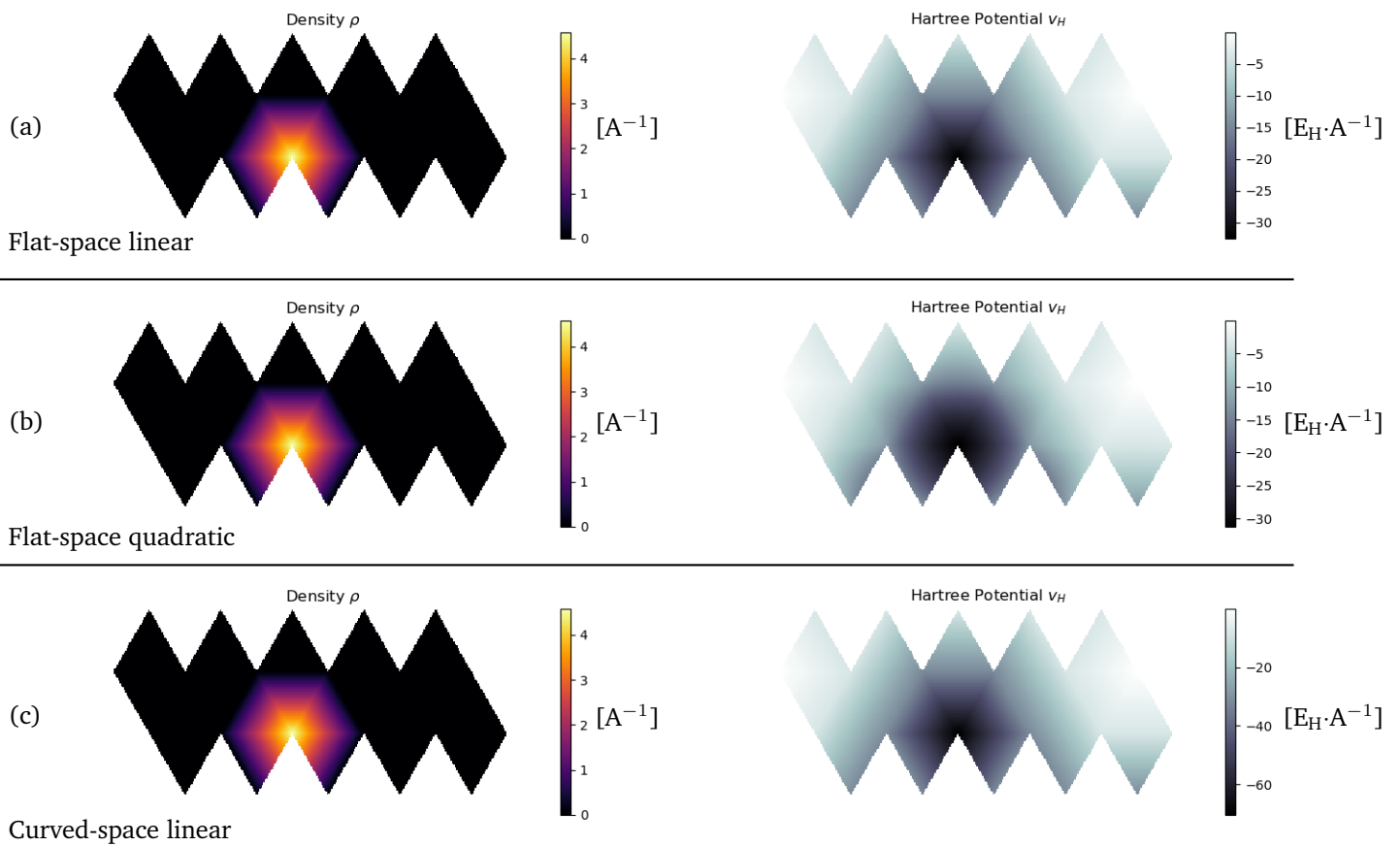
Figure 4.7 (a) and (b) which are the flat space linear and quadratic basis, show an almost identical response in distribution and strength. However the visualization clearly shows the coarse nature of the linear implementation. For the curved implementation in figure 4.7c the solution is similar. The potential differs quite noticeable in strength compared to (a) and (b).

Although this seems strange it is not unexpected. Recalling the mathematical approach of the cotangent laplacian, this included scaling with respect to the Voronoi cell for each triangle vertex. The use of dividing by Voronoi cells to scale the cotangent expression in equation 3.1.29 is a popular approach but not unique mentioned in section 3.1.5. We should therefore not necessarily expect the curved and linear approaches to yield in this case potentials with corresponding strengths.

One could investigate a possible scaling between the flat and curved space implementations. This could be done by constructing a dual mesh arising only from hexagons and scale the curved space laplacian to yield the same result as the flat space solution, since such a mesh has Gaussian curvature of 0 everywhere. A future task is therefore a matter of scaling the solution of the potential to an appropriate strength in the DFT.

<sup>5</sup>If a model only accounted for the hartree potential this shift is not necessary, since points with the lowest repulsion would create the same physical behaviour if defined to attract.





**Figure 4.7.:** Solutions to the Hartree potential arising from the densities from which they arose. (a) and (b) are linear and quadratic flat space implementations while accounts for curvature (c). All densities shown left are of an localized nature which yields a Hartree potential peaking in this localization.

### Symmetric Density on $C_{20}-I_h$ Surface

Now we turn to a solution in which the physical inaccuracy is apparent due to the coarseness of the implemented mesh. Figure 4.8 show the densities investigated, with a density localized on 10 triangles with the distribution peaking in the bottom and top vertices. The computed potentials are inaccurate approximations to a correct continuous solution since it fails to capture several aspects of a correct continuous solution.

If we turn to the linear solutions in (a) and (c) the behaviour of the potential is to drop when moving away from the vertices with the localized distribution. It however reaches its minimum and is constant across all triangles in between the triangles with non zero density. We would expect it to decrease further as we approach the furthest distance away from the concentrated density distributions. A true solution will not be a horizontal line in the middle of constant magnitude since distances between the localized densities are not constant along this line<sup>6</sup>. It can however be helpful to think of when investigating the solutions since the potential variations on such a line can be rather small. While physically wrong, the solutions discussed is a product of a coarse mesh and a basis of low polynomial order. The linear elements have no vertices where the minimum occurs and can simply not represent this behaviour any better.

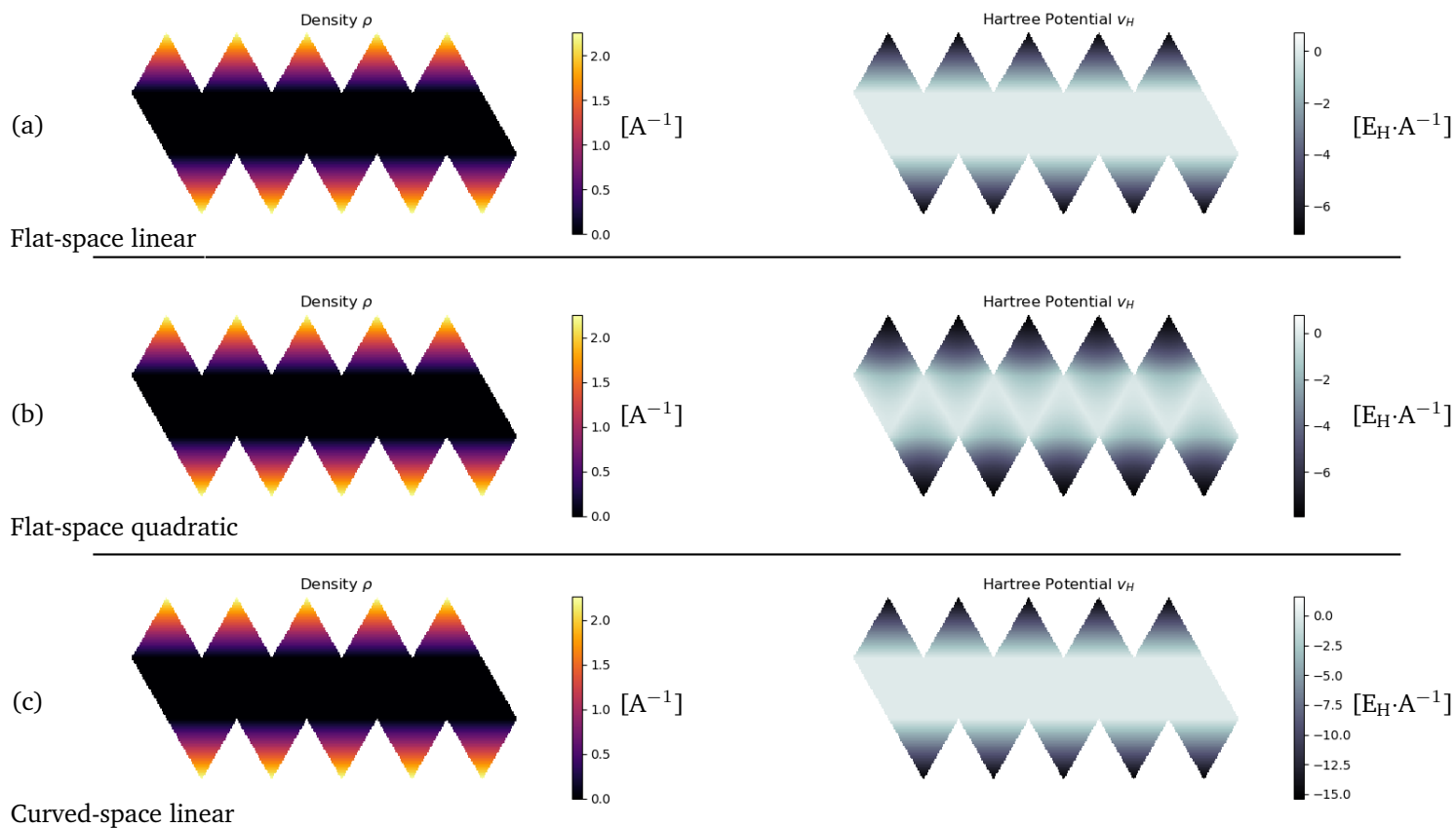
Reaching a better solution with the same polynomial order can only be done by using a finer mesh<sup>7</sup>. Solutions after further dividing all triangles into four smaller triangles are shown in figure 4.9. This shows a more smooth gradient behaviour of the potential as we are moving away from the localized charge distribution. The minimum occurs where it is expected to, but this is only made possible due to nodes being present at the minimum. This showcases the importance of the mesh. Note all triangle nodes on the horizontal middle line will be the same distance from the densities. There can therefore be no variation in the potential along this line.

The quadratic solution in figure 4.8 just as in the linear case fails to capture the necessary physical behaviour. The nodes that are on the before mentioned horizontal line does have the smallest value, but is only slightly different than the vertex nodes for the same triangle. The finer mesh in figure 4.9b yields a more physical sound solution. While difficult to see in figure 4.9b the solutions on the horizontal line vary slightly. The minimum value of 0 occurs on the triangle vertices on the line, with the mid point node having values of  $\approx -0.155 E_H \cdot A^{-1}$ . Note the smoothness of this solution compared to the linear implementations in figure 4.9 (a) and (c).

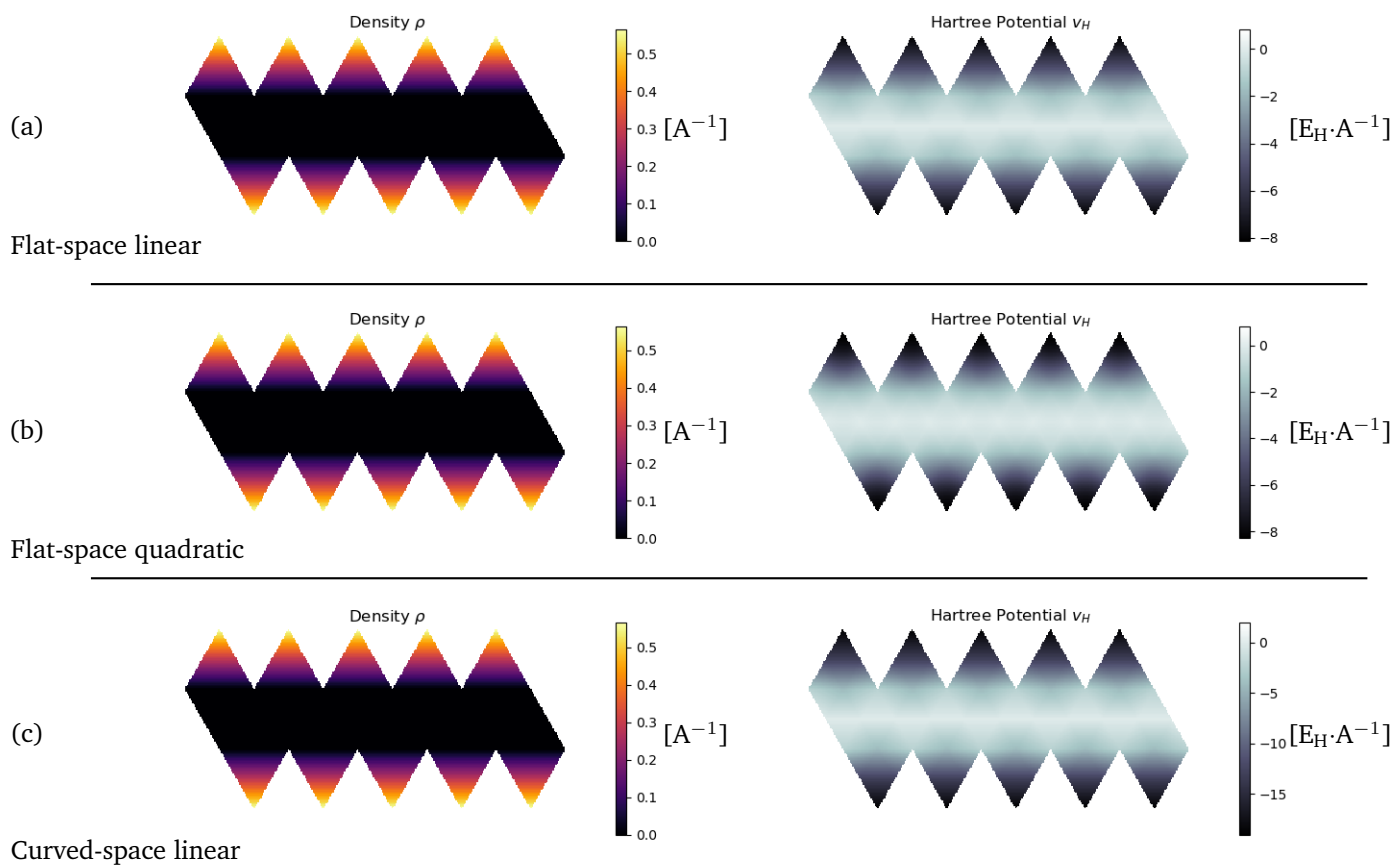
---

<sup>6</sup>This would be the case for a sphere with densities located at two opposite poles.

<sup>7</sup>assuming the original mesh is sufficiently regular



**Figure 4.8.:** Inaccurate solution to the Hartree potential arising from the densities from which they arose due to the coarse mesh. (a) and (b) are linear and quadratic flat space implementations while accounts for curvature (c). All densities shown left are of an localized nature which yields a Hartree potential peaking in this localization.



**Figure 4.9.:** A finer mesh applied to the Hartree solutions in figure 4.8. (a) and (b) are linear and quadratic flat space implementations while accounts for curvature (c). All densities shown left are of an localized nature which yields a Hartree potential peaking in this localization.

## General Considerations

A problem for the system at hand is however the periodic boundary conditions. In a general FEM setting the approach is to use Dirichlet boundary conditions are usually added, i.e. the solution is set to zero sufficiently far away from the region of interest. This is the case in where [25] this is enforced both on the electrostatic potential and the electronic wavefunctions. Due to the periodic boundary no values can be enforced, this leaves Poisson's equation undetermined. Poisson's equation in periodic boundary conditions assure a unique set of solutions where two solutions differ by an additive constant. The only consideration of this additive constant within this work is that all potentials are shifted to have a maximum of value of  $0 \text{ E}_H \cdot \text{A}^{-1}$  at the point of least repulsion. An alternative approach to this is needed, since the system with correctly scaled nuclei interactions must respond accordingly with a correctly scaled electronic behaviour.

The focus will throughout the rest of this thesis be on the linear curved space implementation. It not only captures the true nature of the surfaces better, we avoid an apparent problem with the quadratic representation.

In the SCF loop a new density is a linear combination of single particle densities, the densities will therefore always be positive as it should be. However, the quadratic basis may have negative values in the polynomial representation between the nodes. Take the vector  $f_i = \int_{\Omega} \rho(\mathbf{x}) \varphi_i(\mathbf{x}) d\mathbf{x}$ , when computing this the Gaussian quadrature method need a polynomial representation at certain coordinates, where a negative density may occur. An automated warning was set up in SCF loop to identify if this happened. This was not an unlikely occurrence and the focus is therefore shifted to the linear curved space implementation for now.














l:		$P_\ell^m(\cos\theta) \cos(m\varphi)$	$P_\ell^{ m }(\cos\theta) \sin( m \varphi)$
0	s		
1	p		
2	d		
3	f		
4			
5			
6			
	m:	6 5 4 3 2 1 0	-1 -2 -3 -4 -5 -6

Figure 4.10.: The spherical harmonics yielding the angular part of the hydrogen orbitals. Downloaded from wikipedia [26]

## 4.2.2 Kohn-Sham Orbitals

The results discussed so far have been based on whether or not solutions using the FEM software yield physically sound behaviour from intuition. Having results from previous similar simulations to compare with would obviously be ideal and aid in further implementing and tweaking of the FEM software<sup>8</sup>. However now that we turn the attention to the eigenvalue problem of the Kohn-Sham equations, solutions are shown which can be seen as a approximations to a well-known system with familiar solutions.

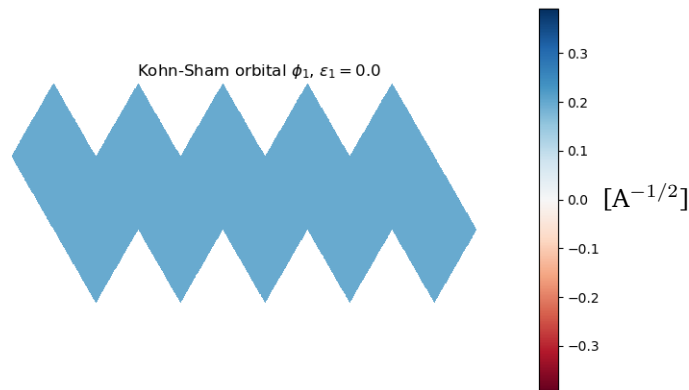
The eigenvector solutions presented will once again be on the  $C_{20}-I_h$  surface using the contangent Laplacian. Constructing an environment with a uniform distribution of density yield a constant potential which is fed to the Kohn-Sham solver. Here we will investigate solutions based on the fact that the  $C_{20}-I_h$  surface can be seen as a crude polyhedral approximation to represent a sphere.

As we know the quantum mechanical solutions of the hydrogen atom are electron wave functions occurring due to a stationary positively charged proton in the center. The electro-static potential arising from the presence of a proton is only dependent on the distance from the proton, thus the potential has perfect spherical symmetry. Our 2-dimensional surface with a constant potential can therefore be viewed as a sphere enclosing a charge. Our solutions to the Kohn-Sham equations must consequently somewhat resemble the spherical harmonic nature of the hydrogen wave functions illustrated in figure 4.10. The results that follow were obtained using a mesh consisting of  $20 \cdot 4^3 = 1280$  equilateral triangles.

The first 9 Kohn-Sham orbitals computed i.e. the eigenvectors with the lowest eigenvalues are shown in figure 4.11, 4.12 and 4.13. Starting with the lowest energy orbital shown in figure 4.11. This has a constant value and subsequently also a constant density on the surface. This is identical to the s orbital in figure 4.10. It has the lowest energy corresponding to the hydrogen solution as well as being non-degenerate.

The next orbitals in line are three solutions with an identical energy as seen in figure 4.12. All consist of two peaks of a positive and a negative nature. Here the value zero correspond to the plane in between the negative and positive region in each p orbital in figure 4.10. Just as in the hydrogen atom where the non-zero angular momentum break the spherical symmetry, the eigensolver compute the three

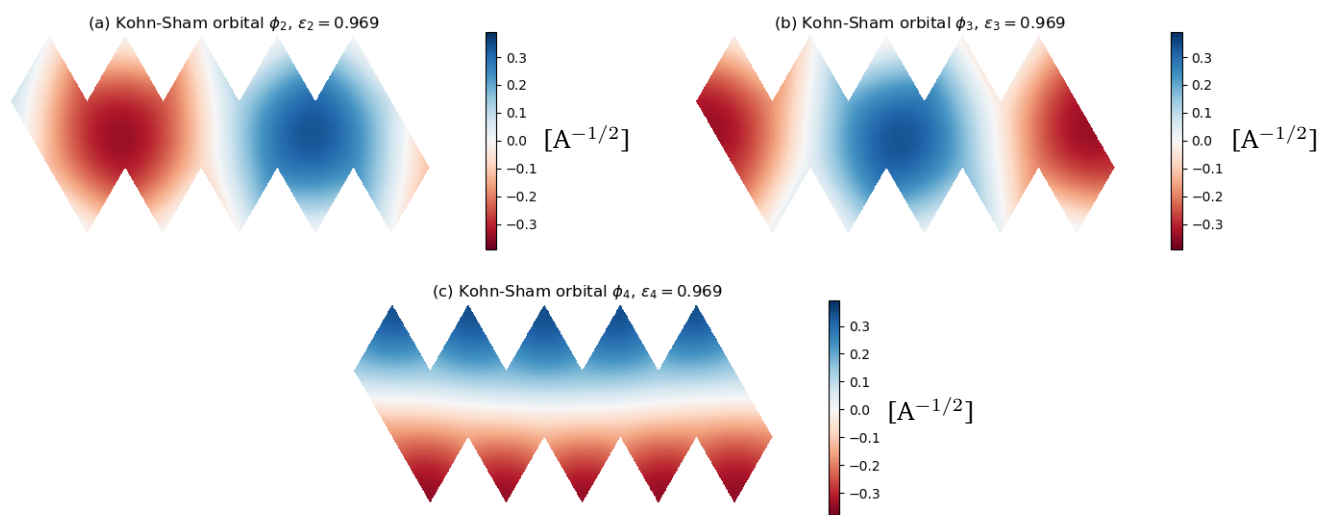
<sup>8</sup>which will mainly be more complex discretizations to the Laplace-Beltrami operator as well as basis functions of higher polynomial order.



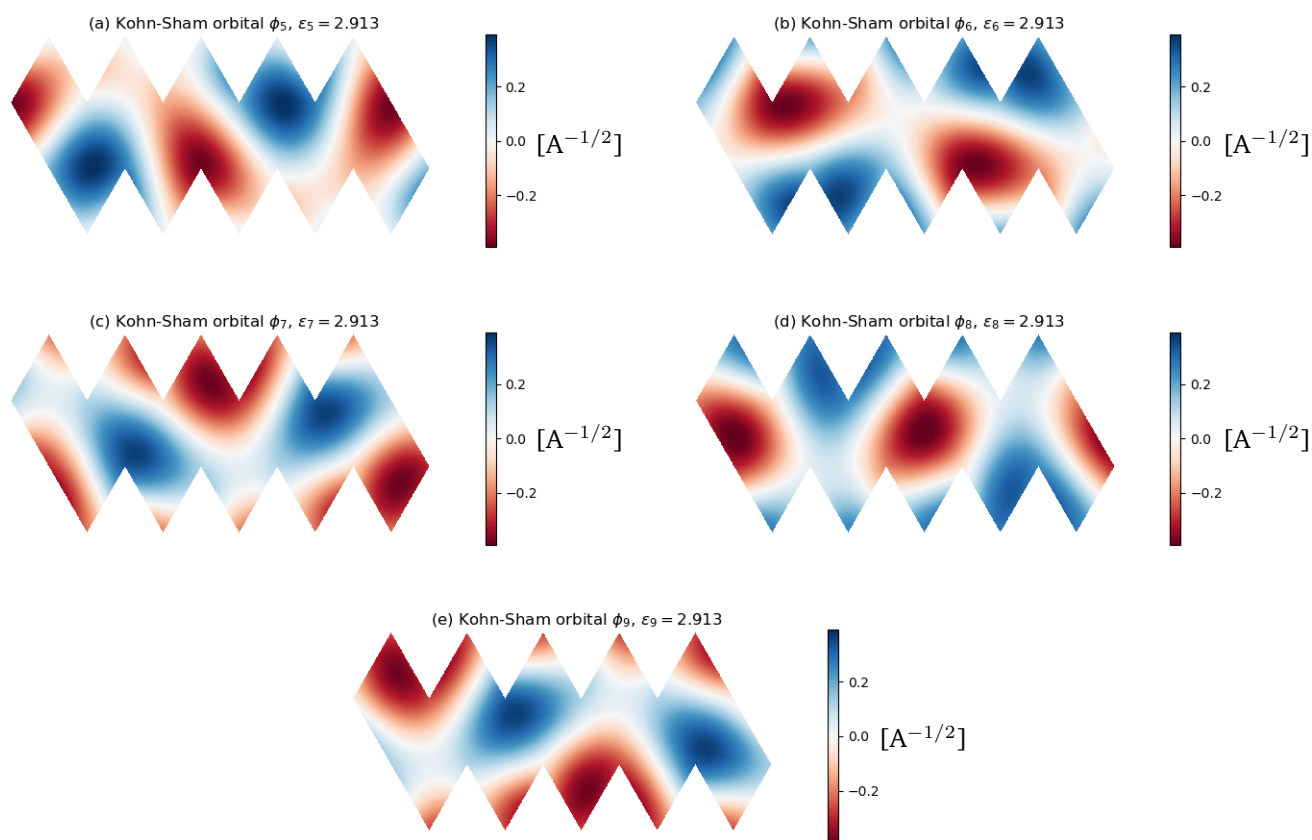
**Figure 4.11.:** The lowest energy orbital solution in the Kohn-Sham equations on the  $C_{20}\text{-}I_h$  surface corresponding to a spherical cut-out of the s orbital in the hydrogen atom. The solution is found using the FEM software with a detailed mesh consisting of 1280 triangles using the cotangent Laplacian. Figure 4.12 and 4.13 are illustrations of higher energy orbitals. Note the energy is in  $E_h$

orthogonal eigenvectors spanning the eigenspace yielding the p orbitals. The axes the solutions picks out in figure 4.12 is easily comparable to the hydrogen three time degenerate p orbitals where the magnetic quantum number is  $m = 1$  for (a),  $m = -1$  for (b) and  $m = 0$  for (c).

After the p like orbitals we have five eigenvectors that all have the same energy shown in figure 4.13. This is once again corresponding with the hydrogen solutions in this case the d orbitals. The complexity of the orbitals does make the manifold unfolding much harder to interpret. If we take the special case of  $m = 0$  where we would expect two distinct circular peaks of the same sign opposite of each-other. This is the case for figure 4.13 (d), which shows two negative regions surrounded by a positive band. The other orbitals show two distinct peaks in both the negative and positive values in line the the d orbitals with  $m \neq 0$ .



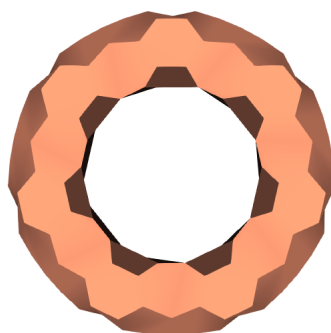
**Figure 4.12.:** Kohn-Sham orbitals number 2, 3 and 4 if ordered after lowest eigenvalue for the  $C_{20}-I_h$  surface. The eigenvectors correspond to p orbitals which pick out three unique axes in the system. They all have the exact same eigenvalue and are identical differing by orientation. Note the energies are in  $E_h$



**Figure 4.13.:** Kohn-Sham orbitals number 5, 6, 7, 8 and 9 if ordered after lowest eigenvalue for the  $C_{20}-I_h$  surface. All energies which are in  $E_h$  are identical and while the eigenvectors correspond to d orbitals in the hydrogen atom.



0



**Figure 4.14.:** Kohn-Sham orbital  $\phi_1$  with an energy of  $\epsilon_1 = 1.000 E_H$

### Kohn-Sham Orbitals on $C_{168}$ Torus

To emphasize the finite element PDE solvers applicability beyond fullerenes we turn the attention to the case of the  $C_{168}$  torus constructed in section 3.6. The FEM has been built to work for all dual representations arising from a cubic graph with periodic boundary conditions. While the triangulation for the cases of fulleroids and fullerenes are embedded in a real space representation it is not a necessary restriction set by the software.

Just as in the above  $C_{20}-I_h$  surface the presented results will be of the orbitals arising on the torus surface due to a constant potential everywhere. The solutions were computed using the dual but will be represented on the real fullerene pentagon/hexagon representation. Each polygon will have a constant colour corresponding to the value of the dual node in the middle. The solutions were computed using the original number of triangular cells in the dual of 168, with  $N_{dof} = 84$ .

The orbital of lowest energy shown in figure 4.14 is non degenerate. It is perhaps a bit unclear due to the shading, but it is has constant nodal values everywhere. The solutions then interestingly exhibit solutions of Kohn-Sham orbitals are 2-fold degenerate. All the pairs of orbitals in figure 4.15, 4.16 and 4.17 show a somewhat continuous behavior. The true orbitals must be continuous and the computed orbitals should obviously obey this. There is of course a limitation to the complexity of the solution the linear implementation can represent, which is why we are only concerned with the low energy orbitals in this case. No orbitals jump from a maximum value to a minimum value in discontinuously, i.e. there is a gradual difference in color between a maximum and minimum value.

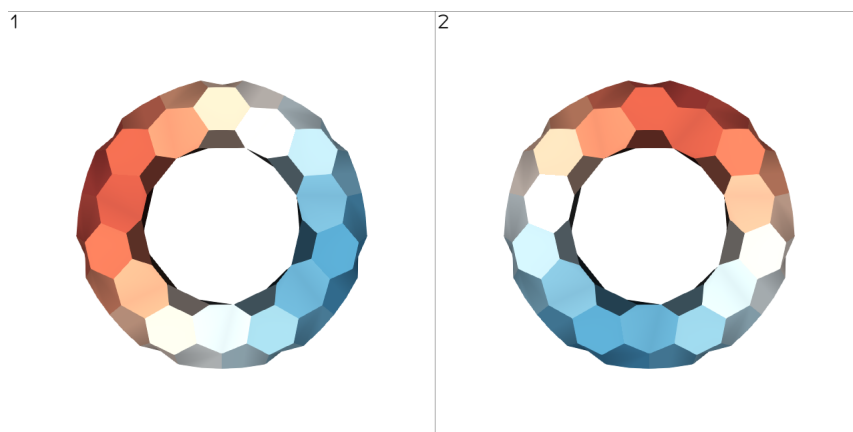


Figure 4.15.: Degenerate Kohn-Sham orbitals  $\phi_2$  and  $\phi_3$  with an energy of  $\varepsilon = 1.011 E_H$

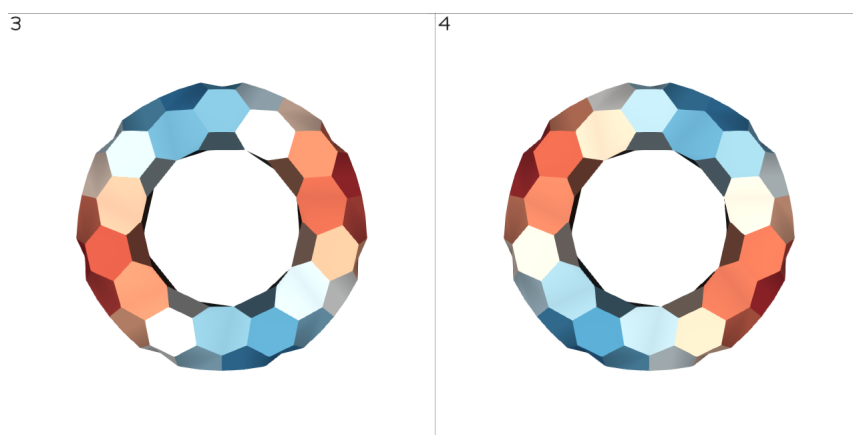


Figure 4.16.: Degenerate Kohn-Sham orbitals  $\phi_4$  and  $\phi_5$  with an energy of  $\varepsilon = 1.043 E_H$

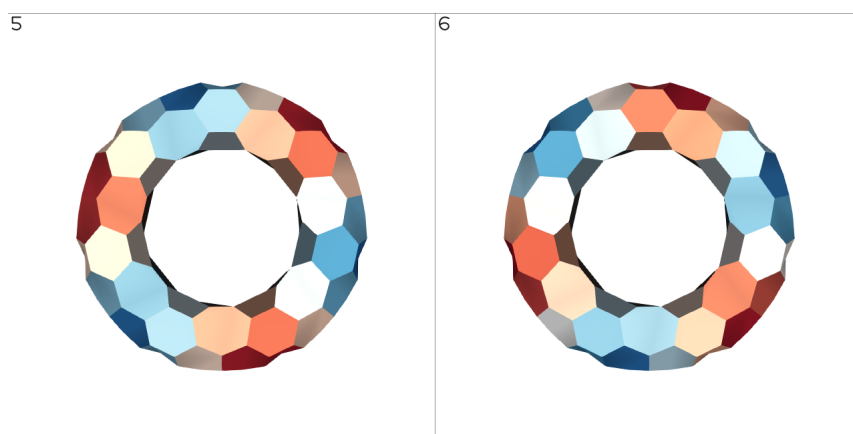
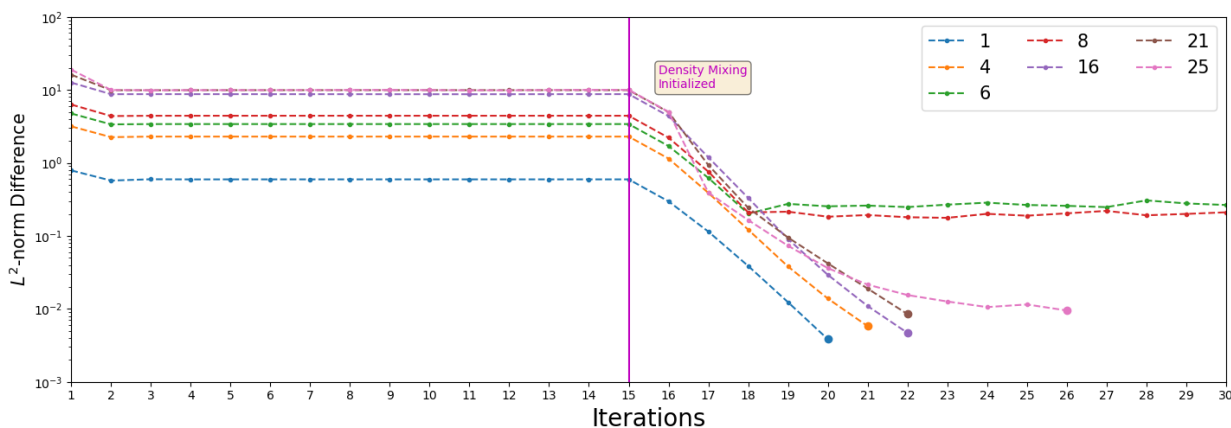


Figure 4.17.: Degenerate Kohn-Sham orbitals  $\phi_4$  and  $\phi_5$  with an energy of  $\varepsilon = 1.095 E_H$



**Figure 4.18.:** The  $L^2$ -norm difference between  $\rho_i$  and  $\rho_{i+1}$  in the SCF loop. Here the number in the legend denote the number of orbitals of which the density consist i.e. how many electrons present in the system. The SCF loop initializes the density mixing after 15 iterations which allow several of the simulations to converge to the  $10^{-2}$  criteria used.

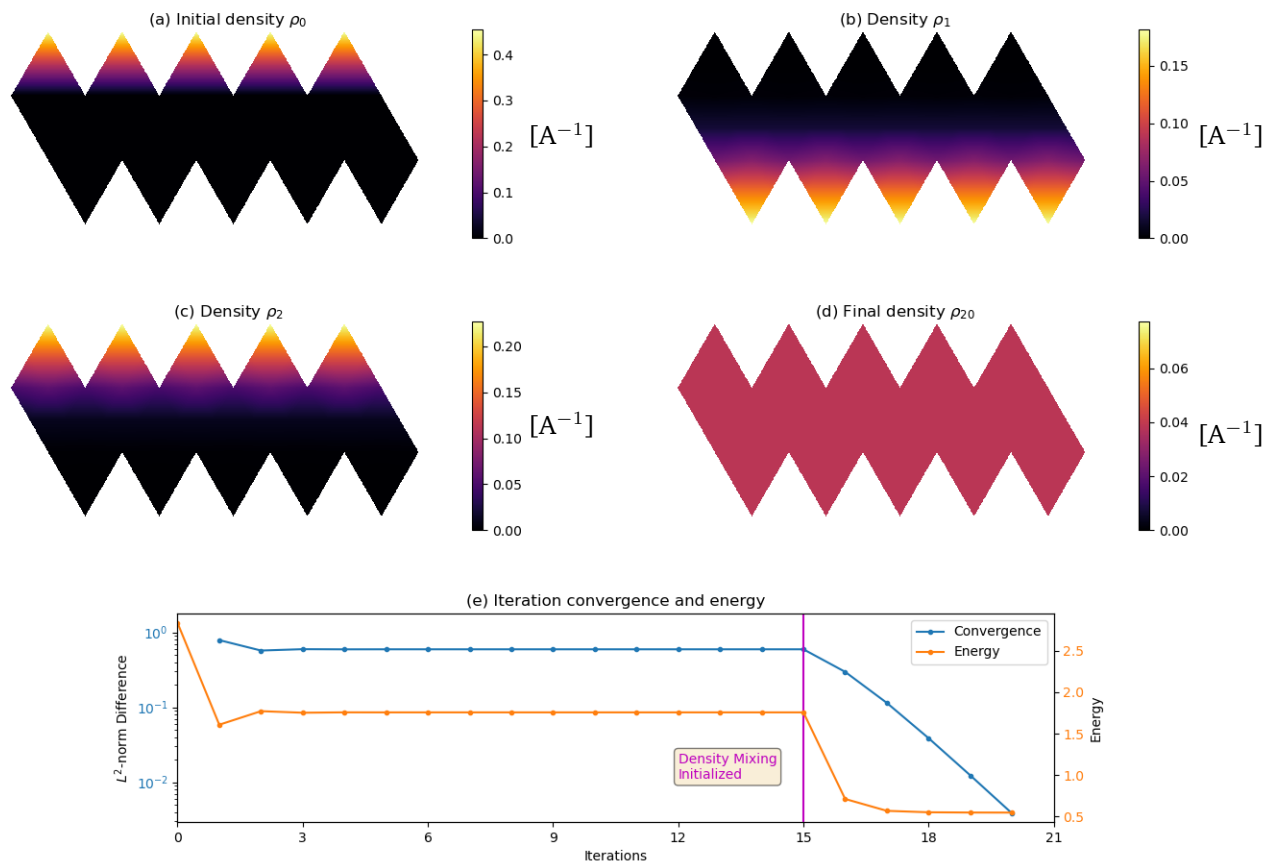
### 4.2.3 SCF Loop

While a full 2-dimensional DFT with nuclear attraction is beyond the scope of this project, a SCF loop algorithm was constructed where the Hartree potential and Kohn-Sham equations are obtained using the developed PDE solver. The SCF loop implementation will however tested in systems with only electrons present. An initial localized density will be the initial density for simulations with a varying number of electrons. It will then be investigated what systems converge as well as the density they converge to. The  $C_{20}$ - $I_h$  surface will serve for discussion of general behaviour, examples of a few simulations will then be given for the surfaces of  $C_{60}$  nanotube and a  $C_{120}$ - $D_6$ .

Starting with figure 4.18 show the convergence of the density at iteration  $i$  for a few selected simulations with a various number of electrons on the  $C_{20}$ - $I_h$  surface. All simulations where performed with a mesh consisting of 80 elements and a convergence criteria of 0.01. At iteration number 15 the density mixing scheme is initialized and used throughout to highlight it's importance and behaviour for minimizing charge sloshing. The convergence behaviour of the all the simulations using from 1 to 24 orbitals are included in SCF loop can be found in appendix A. The following sections will be closing in on specific simulations in figure 4.18.

#### 1-Electron Density on the surface manifold of $C_{20}$ - $I_h$

The 1-electron simulations initial density can be seen in figure 4.19a. This localized density distribution will inevitably create orbitals mainly with distribution opposite of this. The next density computed based on  $\rho_0$  is seen in figure 4.19b. This will be an endless loop of charge sloshing with c showcasing the density for the next iteration. This distribution would bounce back and forth iteration after iteration and partly explain why the initial density can be extremely influential. We see in figure 4.19e that the energy of the states during the charge sloshing are identical. It is not until density mixing is introduced that the densities start to converge to a more stationary structure between each iteration. While the convergence of the density is computationally an easily constructed parameter to control the SCF-loop, it would bare no physical significant if the energy did not minimize as well. This energy minimize to a evenly density distribution which is seen in figure 4.19. Note that the convergence has



**Figure 4.19.:** SCF simulation of  $C_{20}-I_h$  with a density constructed from a single orbital. The SCF loop is initiated with the density seen in (a) while (b) and (c) show the first and second computed densities respectively. (e) Show the  $L^2$ -norm difference between density  $\rho_i$  and  $\rho_{i+1}$  and the energy in  $E_h$  associated with density  $\rho_i$ . The convergence criteria of 0.01 is met 5 iterations after density mixing is introduced and yields a uniform final density shown in (d). Convergence criteria 0.01 using a mesh of 80 triangles, introducing density mixing after 5 iterations.

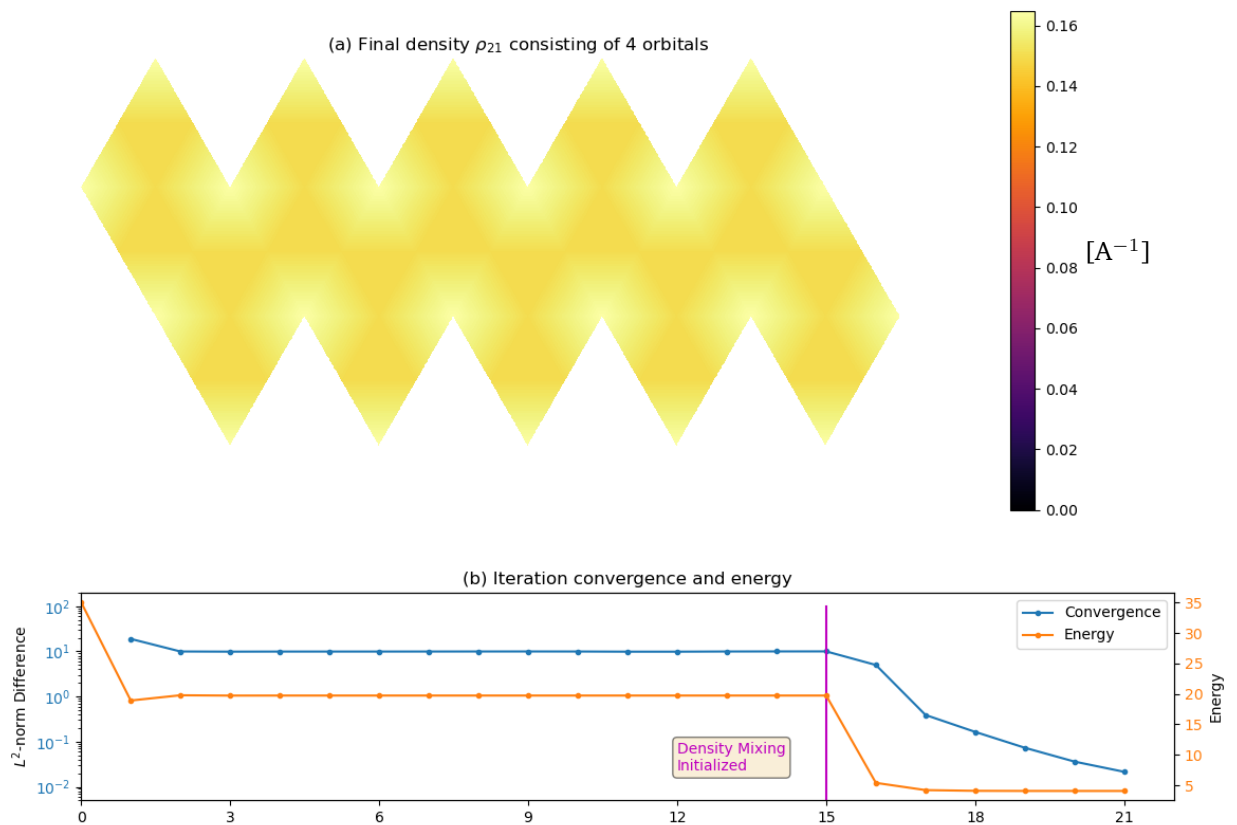
nodal values  $\approx 0.0385 A^{-1}$ , which is easily shown to confirm the normalization for a constant density  $\int_{\Omega} \rho_0 dx = \rho_0 \int_{\Omega} dx = 0.0385 A^{-1} \cdot 20 \frac{\sqrt{27}}{4} A \approx 1$ .<sup>9</sup>

#### Many-Electron Density on the surface manifold of $C_{20}-I_h$

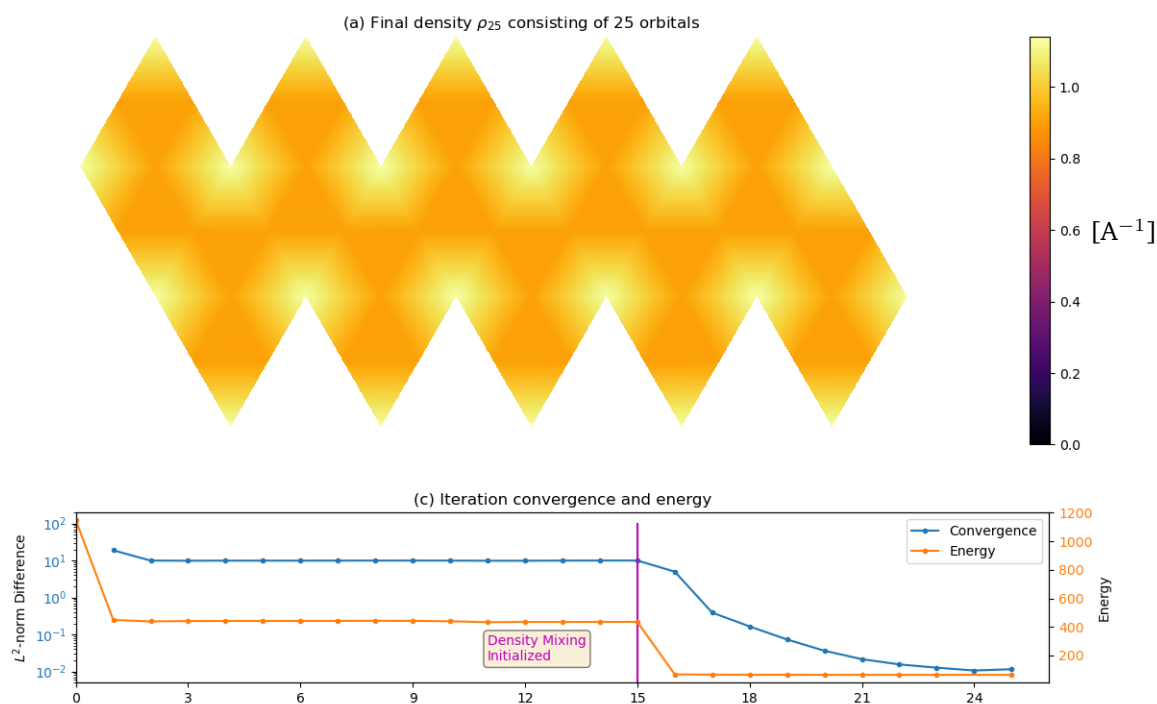
A 4-electron system an very similar convergence and energy behaviour to the one electron system can be seen in figure 4.20b. The final density reached shown in figure 4.20a is almost a uniform distribution. The distribution does however contain small peaks, which are located at the vertices whose Voronoi element is hexagonal. This distribution will yield a potential which is almost flat, though with a slight repulsion at the small density peaks. It is due to the repulsive nature being small, the interplay between the Kohn-Sham orbitals and the density mixing will yield a similar structure, thus the final density also depend on the mixing scheme and  $\alpha$  parameter. This solution is not far from a uniform density and is within 1% in energy.

Another converged simulation is that of 25-electrons which can be seen in figure 4.21. It has a similar density structure and will produce densities alike the one shown due to the same arguments. The relaxed densities are all close to a flat distribution, with small density peaks recognizing the symmetry within the manifold. This therefore greatly exemplifies the dependence on the number of orbitals the

<sup>9</sup>since the system has 20 triangles each of  $A = \frac{\sqrt{27}}{4}$ .



**Figure 4.20.:** (a) Final density for the SCF loop for 4-electron system with the energy in  $E_h$  and convergence behaviour plotted in (b). The simulation was performed with an initial localized density causing charge sloshing, which is inhibited by introducing density mixing.



**Figure 4.21.:** (a) Final density for the SCF loop for the 25-electron system with the energy in  $E_h$  and convergence behaviour plotted in (b). The simulation was performed with an initial localized density causing charge sloshing, which is inhibited by introducing density mixing.

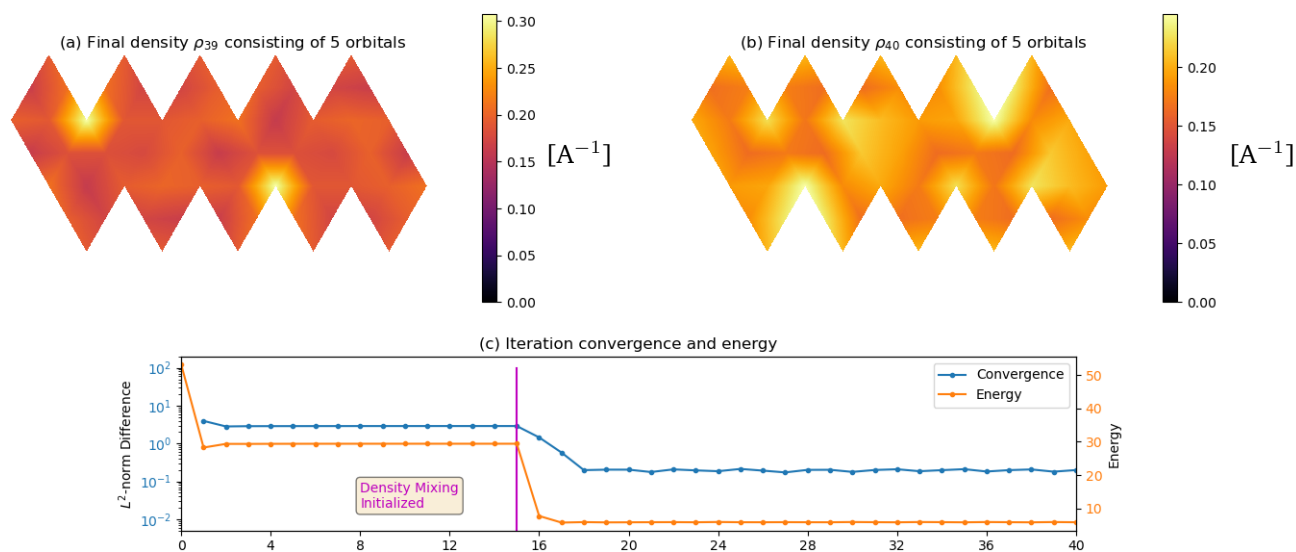
density is constructed from. In figure 4.22 an example of a non-converge simulation is shown. Here the somewhat chaotic density (a) yields (b), which show a high spatial displacement of charge. The densities, although not far from flat, look nothing alike and will show no converging behaviour if continued further. The density mixing does have an effect, but it effectively only shifts the value of the  $L^2$ -norm seen in (c), and it iterates until manually stopped. The energy remains alike in the system just like  $L^2$ -norm, meaning that while the difference in densities is apparent it is not for the energy. The simulations are quite delicately dependent on the mixing scheme, initial density and number of orbitals included.

### Many-Electron Density on the Surface Manifold of $C_{60}$ -nanotube

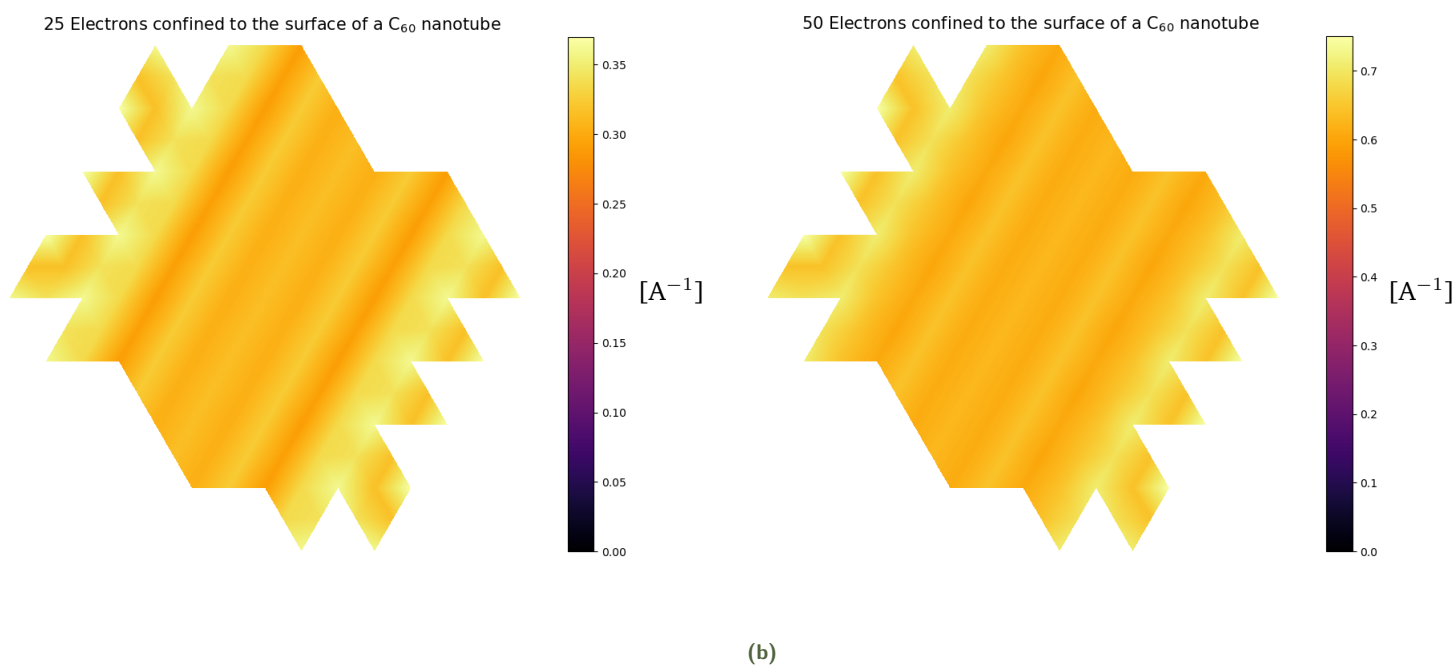
Now we apply the SCF loop to a larger fullerene than the dodecahedrane namely a  $C_{60}$  nanotube with the already familiar unfolding illustrated in figure 4.4. Simulating 25-electron and 50-electron densities yield the final densities shown in figure 4.23. Here the densities look quite similar and clearly converge to a density that highlights the symmetry in the nanotube. This seems to be the case for all converging densities the SCF loop computes for the electrons contained on the manifolds.

### Many-Electron Density on the surface manifold of $C_{120}$ - $D_6$

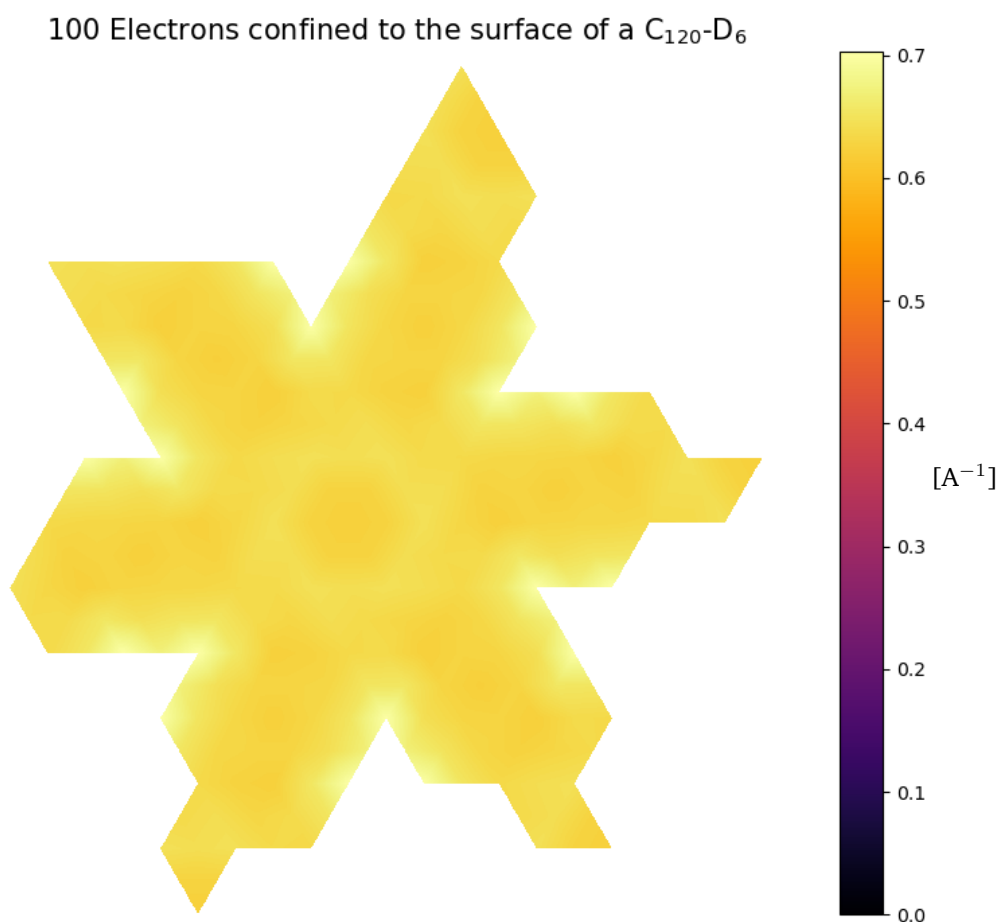
Applying the SCF loop to an even larger fullerene surface from a  $C_{120}$ - $D_6$  molecule. The unfolding for this surface is seen in figure 4.25 since this is the first encounter of the unfolding. A SCF loop simulation yielding the final 100-electron density can be seen in figure 4.24. Except the small peaks near the pentagons this is almost uniform. An outline of a hexagon can though vaguely be seen in the middle. All of these vertices have an equal geodesic to the closest pentagon dual node outlining the symmetry. It must be emphasized that these densities are the optimal result that SCF loop can reach. The method is



**Figure 4.22.:** The failed SCF loop for the 5-electron simulation. (c) The convergence does not decrease much after the density mixing is introduced and it remains flat. After 39 iterations the density (a) yield the wildly different density (b). Energy in in  $E_h$ .



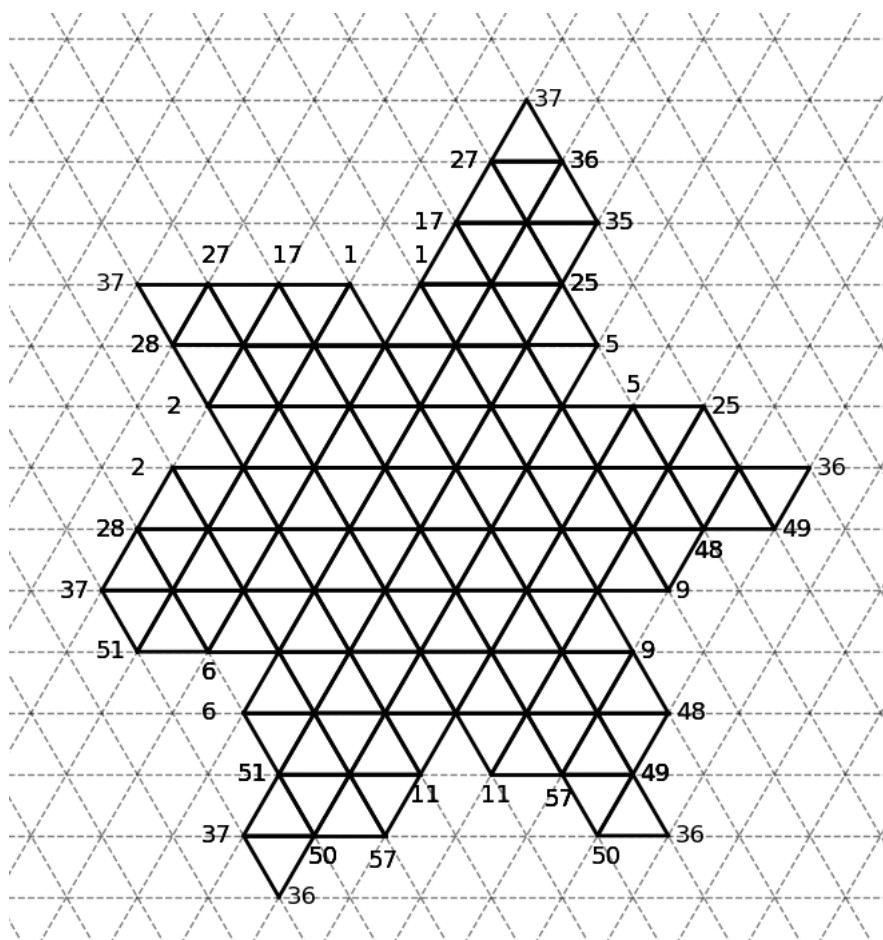
**Figure 4.23.:** Final converged electronic densities on a  $C_{60}$  nanotube surface. The 25-electron density in (a) and 50-electron density in (b) have both reached a convergence criteria of  $10^{-6}$  in the SCF loop after 22 and 24 iterations respectively. The density initial density was a flat distribution and the mixing scheme was initiated after 5 iterations



**Figure 4.24.:** Final converged electronic densities on a  $C_{120}-D_6$  surface. The 100-electron density reached a convergence criteria of  $10^{-6}$  in the SCF loop after 15 iterations respectively. The density initial density was a flat distribution and the mixing scheme was initiated after 5 iterations. The original 120 triangle dual mesh was further refined to 480 triangles.

especially interesting due to future aspect. The SCF scheme in itself is not an approach especially suited to the nuclei-less 2-dimensional surfaces





**Figure 4.25.:** A  $C_{120}\text{-}D_6$  dual unfolding to the Eisenstein plane, where all vertices that appear more than once in the unfolding are labelled. Connecting these matching labels will yield the 3-dimensional dual structure of the molecule.

Now with all the the details that have been presented in this work, let us take a step back and walk-through the bullet points of the work and challenges the research project faces in the future. This will be a collection of tasks that are generally beyond the scope of this thesis as well as modifications needed to the written software.

## 5.1 FEM Related Challenges

One of the important aspects of the FEM software are the basis function implementations. While the linear curved space implementation is easy to work with, the complexity which it captures is decided only by the mesh. It seems no clear consensus within the engineering and computer science communities on how to approach non linear basis functions that account for curvature, is present. An approach to this, would allow for the possibility of using high order polynomial functions and therefore more complex simulations.

The 2-dimensional approach was based in the hope that the extreme electron mobility along the graphene like surfaces dominate over electronic interactions through the hollow volume. While this may generally be the case for many of the hexagons, the pentagons induce positive Gaussian curvature and will make the inter volume interactions more likely. Not only does this lose its validity at the 12 pentagons, but there are the areas of greatest interest. Here the idea is to extend the surface mesh to a simplicial complex, consisting of 2-dimensional triangles and 3-dimensional tetrahedra. These 3-dimensional shapes will then be constructed at areas of interest near the pentagons. This is in fact the subject of a thesis currently within the group.

## 5.2 DFT Related Challenges

As mentioned we have an undetermined solution to Poisson's equation on periodic boundary conditions. This means that a solution to Poisson's equation only define the Hartree potential up to an additive constant. The Hartree potential must represent the electronic repulsion shifted correctly, before the nuclei are accounted for. An idea to investigate this shift would be to construct a simulation in which the Hartree potential or its corresponding energy to a given density is known.

Once this is achieved the attention can be turned to the positively charged nuclei. In the dual representation each atom is situated at the center of a face. This must be approached in a smart manner, due to finite element methods inability to generally represent functions with exponential behaviour, present due to atomic cusps of wavefunction and densities at nuclear centers. To even represent this in a FEM sense, the mesh would need to become finer and finer towards the cores, until itself is infinitely dense.

However since it is mainly the valence electrons that are of our interest we are aided by the concept of pseudopotentials. Pseudopotentials attempts to encapsulates the joint potential behaviour of the core and the inner shell electrons.

Another new untested approach would be to add the graphene solution to the 2-dimensional unfolding. This is idea is that a DFT treatment of the fullerenes can be developed for which solutions are difference densities to the graphene-like solutions, smooth enough to be treated numerically with finite elements. The part of the solution that looks like graphene wrapped around the polyhedron is also the same for all isomers. This can therefore hopefully be factored out of the solutions and represented as a mostly-smooth potential. This idea is currently in an early speculative state, but an interesting approach none the less.

The DFT model constructed does not account for spin in any way. Two different ways of approaching this is through the restricted and unrestricted formalism. The restricted formalism enforce that up and down spins accompany each-other and doubly occupy orbitals. Contrary to this, is the unrestricted formalism in which a "pair" of up and down spins can occupy different spatial orbitals with energies sufficiently close to each other. Whichever and however these are implemented, we compute a density as the sum of the spin densities  $\rho(\mathbf{x}) = \rho_{\uparrow}(\mathbf{x}) + \rho_{\downarrow}(\mathbf{x})$ . Extending the used LDA for the exchange potential in a spin-polarized system is straight forward. This is however not the case for the implemented correlation. The desire to implement a generalized gradient approximation has however been discussed in the group. It could therefore practical to tackle these challenges simultaneously.

If the 2-dimensional approach manages to approximate true density appropriately to search through large isomer spaces. A search of specific characteristics will then perhaps yield 1000-100,000 isomer candidates. The plan is then to produce the 3-dimensional structures of the specific isomers using force field optimization from the graph, recently researched within the group[27]. Projecting the surface densities into a 3-dimensional representation will allow for deriving molecular properties and finally map the isomers which are actually interesting to perform a complex time consuming DFT analysis on. The early work done in [28] within the group investigated how one might approach this 3-dimensional projection. This was based on testing the hypothesis of whether or not a 3-dimensional density can be approximated by a 2-dimensional representation by

$$\rho(\mathbf{s}, z) = a(\mathbf{s})e^{-b(\mathbf{s})|z|} \quad (5.2.1)$$

Here the  $\rho(\mathbf{s}, z)$  is density dependent on the surface coordinate  $\mathbf{s}$  and the orthogonal distance from the surface  $z$ . Each surface point will then be assigned values of the scalar field functions  $a(\mathbf{s})$  which describe the density on the surface and  $b(\mathbf{s})$  describing the decay away from a certain point on the surface.. The work was somewhat inconclusive with respect to this problem but if future work deem it usable, it allows for simple 3-dimensional projections of the 2-dimensional densities. It was suggested that two different models were needed for the decay inward and outward in respect to the fullerene surface.

## Conclusion

This thesis has constructed an approach to solve partial differential equations on the non-euclidean 2-dimensional surfaces of fullerenes as well as implementing the approach in a preliminary 2-dimensional density functional theory method. The methods developed in this thesis have laid the groundwork for research to come investigating purely 2-dimensional electronic densities on the fullerene surfaces, hopefully yielding sufficient approximations to the true 3-dimensional electronic densities. If successful, this will in theory greatly reduce the computations needed for approximating electronic densities, allowing for us to search for specific molecular properties in huge isomer spaces.

The approach uses a finite element method with a curved space Laplacian-Beltrami operator, which due to the fullerenes geometrical properties, allow for solutions to be obtained by only providing the fullerenes bond graph representation as the input. PDEs on fullerene surfaces are therefore possible without any global coordinates. This was constructed from scratch to allow for full computational control of the method and made it possible to tailor the method to the graph input. More importantly it allow for tweaking the approach to the Laplace-Beltrami operator in the hope of describing the curved 2-dimensional surfaces. This was all implemented in as a python software library, written in a way which allows to easily adapt higher complexity of the polynomial order of basis functions in the finite element method in the future. Once fully functional, it can be be constructed in C++ allowing for much faster computation which can then be included in the Fullerene Library[29].

A 2-dimensional preliminary DFT was then developed with a local density approximation accounting for the exchange-correlation potential. This DFT directly implemented the newly developed surface manifold FEM library to solve Poisson's equation and the Kohn-Sham equations on the fullerene surfaces in an iterative manner within a SCF loop.

Results from the partial differential equations solver was presented. The heat equation was investigated in an qualitative manner on the  $C_{20-I_h}$  and a  $C_{60}$  nanotube surfaces, which yielded meaningful results in terms of verifying that the nodes defining adjacent triangles were computed correctly within the solver. It also yielded an intuitive flow of heat dissipating throughout the fullerene structure converging to a uniform distribution.

Poisson's equation was investigated on the  $C_{20-I_h}$  surface. It showcased the importance of further meshing the dual graph triangulation before solving. This is an important result going forward and the dual triangulation without mesh refinement will need a basis functions of a higher polynomial order. Solutions to the Kohn-Sham orbitals were then presented in the context of the hydrogen atom. The orbitals were computed using a constant potential on the dodecahedron surface. These orbitals were then compared to the solutions of the hydrogen atom and showed to not only obey the same degeneracy present in the spherical harmonics but having analogous orbital shapes. Similar results of orbitals arising from a constant potential along the fullerene surface were presented for a  $C_{168}$  torus, partly to show the PDE solver's general applicability.

Lastly the results for the SCF loop with  $N$ -electrons confined to the 2-dimensional surfaces were presented on the  $C_{20-I_h}$ , a  $C_{60}$  nanotube and a  $C_{120-D_6}$  surfaces. Several simulations for the dodecahedron

surface with a various number of electrons was presented, where several cases yielded converging distribution of the electronic density. The converging electronic density distributions showed to clearly arrange in some symmetric structure within the surface. Converging simulations may therefore generally be easier to obtain on fullerene surfaces within the groups of the highest symmetric order. These results showcased the skeleton for a DFT where the developed PDE solver was applied, hence the intermediate nature of the results.

## Future Work

The work at hand should be considered in the larger context of the Folding Carbon Project <https://www.nbi.dk/~avery/folding-carbon/>. The outlook along with future work presented in section 5 is therefore of great importance. Here extensions necessary to the code developed in this thesis was stated. This also dealt with approaches to include nuclear attraction through pseudopotentials as well as a speculative approach of computing difference densities to the graphene solutions.

If all research within the Fullerene project can successfully realized it could in theory enable researchers and engineers to categorically search huge isomer spaces of fullerenes with desired molecular properties. The specific isomers could then be realized synthetically through rational synthesis. For this to ever be realized a multitude of things has to come together, but the prospects are fascinating none the less.

# Appendices

## A.1 Convergence of C20 Simulations

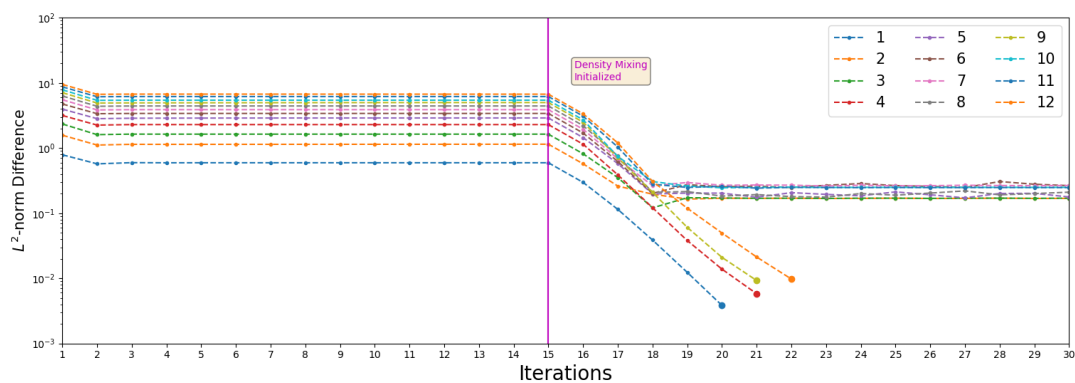


Figure A.1.: The convergence scheme for 1 to 12 electrons confined to the  $C_{20}$ - $I_h$  surface

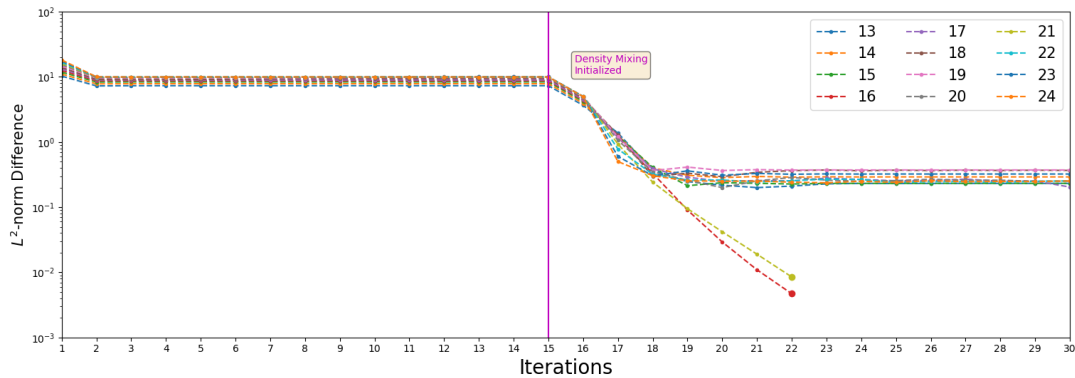
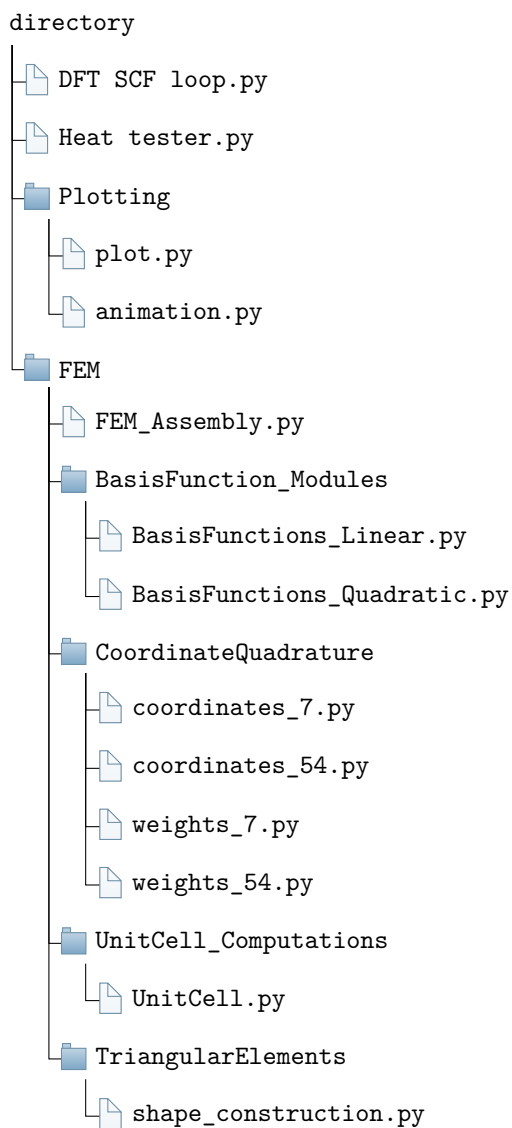


Figure A.2.: The convergence scheme for 13 to 24 electrons confined to the  $C_{20}$ - $I_h$  surface

## B.1 Directory Overview of the Software





# Bibliography



- [1] H. W. Kroto, J. R. Heath, S. C. O'Brien, R. F. Curl, and R. E. Smalley, „C60: Buckminsterfullerene“, *Nature*, vol. 318, no. 6042, pp. 162–163, 1985.
- [2] S. Nam, J. Seo, S. Woo, W. H. Kim, H. Kim, D. D. C. Bradley, and Y. Kim, „Inverted polymer fullerene solar cells exceeding 10% efficiency with poly(2-ethyl-2-oxazoline) nanodots on electron-collecting buffer layers“, *Nature Communications*, vol. 6, no. 8929, 2015.
- [3] J. J. Ryan, H. R. Bateman, A. Stover, G. Gomez, S. K. Norton, W. Zhao, L. B. Schwartz, R. Lenk, and C. L. Kepley, „Fullerene nanomaterials inhibit the allergic response“, *Journal of Immunology*, vol. 179, no. 1, pp. 665–672, 2007.
- [4] O. V. Pupysheva, A. A. Farajian, and B. I. Yakobson, „Fullerene nanocage capacity for hydrogen storage“, *Nano Lett*, vol. 8, no. 3, pp. 767–774, 2008.
- [5] J. Cami, J. Bernard-Salas, and E. P. and S. E. Malek, „Detection of c60 and c70 in a young planetary nebula“, *Science*, vol. 329, no. 5996, pp. 1180–1182, 2010.
- [6] P. Schwerdtfeger, L. N. Wirz, and J. Avery, „The topology of fullerenes“, *WIREs Computational Molecular Science*, vol. 5, no. 1, pp. 96–145, 2015.
- [7] G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot, „House of graphs: A database of interesting graphs“, *Discrete Applied Mathematics*, 161:311–314, 2013.
- [8] L. T. Scott, M. M. Boorum, B. J. McMahon, S. Hagen, J. Mack, J. Blank, H. Wegner, and A. de Meijere, „A rational chemical synthesis of c60“, *Science*, vol. 295, no. 5559, pp. 1500–1503, 2002.
- [9] K. Y. Amsharov and M. Jansen, „A c78 fullerene precursor: toward the direct synthesis of higher fullerenes“, *The Journal of Organic Chemistry*, vol. 73, no. 7, pp. 2931–2934, Apr. 2008.
- [10] K. Amsharov and M. Jansen, „Synthesis of a higher fullerene precursor—an “unrolled c84 fullerene“, *Chem. Commun.*, pp. 2691–2693, 19 2009.
- [11] J. E. Avery, „Wave equations without coordinates i: Fullerenes“, *Rendiconti Lincei. Scienze Fisiche e Naturali*, vol. 29, no. 3, pp. 609–621, Sep. 2018.
- [12] H. Li and H. Zhang, „The isolated-pentagon rule and nice substructures in fullerenes“, *Ars Mathematica Contemporanea*, vol. 15, pp. 487–497, Sep. 2018.
- [13] C. Chuang and B.-Y. Jin, „Hypothetical toroidal, cylindrical, and helical analogs of c60“, *Journal of Molecular Graphics and Modelling*, vol. 28, pp. 220–225, 2009.
- [14] S. C. B. L. R. Scott, *The Mathematical Theory of Finite Element Methods*. Springer Publishing, 1994.
- [15] R. G. P. W. Yang, *Density-Functional Theory of Atoms and Molecules*. Oxford University Press, 1989.
- [16] S. Wandzura and H. Xiao, „Symmetric quadrature rules on a triangle“, *Computers and Mathematics with Application*, vol. 45, no. 12, pp. 1829–1840, 2003.
- [17] W. R. Inc., *Mathematica, Version 12.1*, Champaign, IL, 2020.

- [18] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, „Implicit fairing of irregular meshes using diffusion and curvature flow“, 1999.
- [19] M. M., D. M., S. P., and B. A.H., „Discrete differential-geometry operators for triangulated 2-manifolds“, 2003.
- [20] B. Vallet and B. Lévy, „Spectral geometry processing with manifold harmonics“, *Computer Graphics Forum (Proceedings Eurographics)*, 2008.
- [21] P. M. W. Gill, *Density Functional Theory(DFT), Hartree-Fock (HF) and the Self-consistent Field*. Wiley Online Library, 2002.
- [22] D. M. Ceperley and B. J. Alder, „Ground state of the electron gas by a stochastic method“, *Physical Review Letters*, vol. 45, no. 7, 1980.
- [23] T. Chachiyo, „Communication: Simple and accurate uniform electron gas correlation energy for the full range of densities“, *The Journal of Chemical Physics*, vol. 145, no. 021101, 2016.
- [24] K. Crane, C. Weischedel, and M. Wardetzky, „The heat method for distance computation“, *Commun. ACM*, vol. 60, no. 11, pp. 90–99, Oct. 2017.
- [25] P. Motamarri, M. R. Nowak, K. Leiter, J. Knap, and V. Gaviniya, „Higher-order adaptive finite-element methods for kohn–sham density functional theory“, *Journal of Computational Physics*, vol. 254, pp. 308–343,
- [26] *Alternative picture for the real spherical harmonics*, [https://en.wikipedia.org/wiki/Spherical\\_harmonics#/media/File:Sphericalfunctions.svg](https://en.wikipedia.org/wiki/Spherical_harmonics#/media/File:Sphericalfunctions.svg), Accessed 2020-20-10.
- [27] B. N. Pedersen, „Molecular shapes of fullerenes“, Master’s thesis, University of Copenhagen, Blegdamsvej 17, 2100 København, Oct. 2020.
- [28] K. E. Iversen, „A toolkit for investigating 3d electron densities of molecular surfaces“, Bachelor’s Thesis, University of Copenhagen, Blegdamsvej 17, 2100 København, Sep. 2020.
- [29] P. Schwerdtfeger, L. Wirz, and J. Avery, „Program fullerene: A software package for constructing and analyzing structures of regular fullerenes“, *Journal of Computational Chemistry*, vol. 34, no. 17, pp. 1508–1526, 2013.