

**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

MASTER THESIS

Richard Ivánek

**Two-dimensional model of electron
collisions with molecules**

Institute of Theoretical Physics

Supervisor of the master thesis: doc. RNDr. Karel Houfek, Ph.D.

Study programme: Theoretical physics

Study branch: FTFP

Prague 2022

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date
Author's signature

Many thanks to my supervisor doc. Houfek for all the invested time and effort, to my girlfriend Nikol for her emotional support and to Jakub and Michal for lending me their eyes when I needed them the most.

Title: Two-dimensional model of electron collisions with molecules

Author: Richard Ivánek

Institute: Institute of Theoretical Physics

Supervisor: doc. RNDr. Karel Houfek, Ph.D., Institute of Theoretical Physics

Abstract: In this master thesis we study convergence of selected iterative methods applied to linear systems originating from the finite element method, discrete variable representation and exterior complex scaling in the context of models of electron collisions with molecules. Benchmarks and comparisons with direct methods are made and convergence of methods is analyzed for a one-dimensional two-channel model and a more realistic two-dimensional model of four different molecular systems. Along the way, a review of the mentioned numerical methods and solvers is presented as well as a discussion of the sparsity patterns and numbers of non-zero elements of the used matrix representations. Finally, possible pathologies of preconditioners are shown.

Keywords: quantum mechanics, collisions, numerical solution, iterative methods

Contents

Introduction	3
1 Physical background	5
1.1 A simple two-channel model	6
1.2 Two-dimensional diatomic models	7
2 Mathematical background	13
2.1 Finite element method	13
2.1.1 Discrete variable representation	14
2.1.2 Exterior complex scaling	16
2.2 Solving linear systems	17
2.2.1 Direct methods	17
2.2.2 Iterative methods	18
2.2.3 Preconditioning	18
2.2.4 Kickstart	19
2.3 Sparsity pattern of the left-hand side	19
2.3.1 The two-channel Hamiltonian	20
2.3.2 The two-dimensional Hamiltonian	20
3 Results and benchmarks	27
3.1 The two-channel model	28
3.1.1 Results	28
3.2 The 2D model	34
3.2.1 Convergence properties	34
3.2.2 The cross-section σ^{VE}	43
3.2.3 The cross-section σ^{DA}	43
Conclusion	49
Bibliography	51
A Used grids	53
B Implementation details	55
B.1 Rust Programming Language	55
B.2 Constructing grids	56
B.3 The product grid	58

Introduction

Scattering theory is a valuable tool for measuring particle parameters and for comparing theory with experimental results, because theoretically obtained quantities are straightforwardly translated onto experimental results. Unfortunately, these quantities cannot be obtained analytically for a broad class of problems and for almost none of the realistic models. Numerical methods are therefore a hard requirement to practically utilize the theory.

Various numerical methods like the finite element method have historically been employed to represent differential equations in computers. The resulting matrix systems can then be solved using direct solvers, which, even for sparse systems, result in full matrices and memory often becomes a limitation for more realistic and detailed models, in particular for a higher number of spatial dimensions of a model.

This limitation serves as a motivator to attempt to employ iterative methods instead of direct methods, because they allow the solver implementation to reuse the original matrix, therefore not violating the sparsity of the problem matrix itself. The problem with iterative methods is, that the number of iterations and thus the number of operations to obtain a result with certain precision cannot be easily determined just from the form of the problem and tends to be very large for ill-conditioned systems.

To patch this issue, preconditioners can be used to lower the condition number of a problem while still only working with matrices with similar sparsity patterns as the original matrix. They can lower the number of necessary operations by orders of magnitude and enable the iterative methods to be used in situations where they would normally be unviable.

The magic of preconditioners is however not without its drawbacks, and the solver implementation must make the right choice of preconditioner type to utilize. The optimal choice is not universal for a type of numerical method but rather is distinct for each problem. And there are a lot of different types of preconditioners.

The goal of this work is to build an iterative solver leveraging the finite elements method and benchmark it against selected physical scattering models of electron-diatomic collisions and resonances. Dependence of number of iterations on model parameters is tested and the quality of results is compared with analytic solution when possible. Preconditioners are then taken into account and are compared with each other for a given problem.

Chapter 1

Physical background

This thesis is motivated by problems encountered in the scattering theory. Suppose we have a system of a particle colliding with a target in an initial state $|\psi_0\rangle$ which is an eigenstate of the free Hamiltonian $\hat{H}_0$ with the eigenvalue E . The interaction between the particle and the target is described by the potential $\hat{V}_{\text{int}}$. The solution $|\psi\rangle$ of this scattering problem can be split into two parts

$$|\psi\rangle = |\psi_0\rangle + |\psi_{\text{sc}}\rangle. \quad (1.1)$$

The stationary Schrödinger equation can then be rewritten as

$$\begin{aligned} \hat{H} |\psi\rangle &= E |\psi\rangle, \\ (\hat{H} - E) |\psi_{\text{sc}}\rangle &= -(\hat{H} - E) |\psi_0\rangle, \\ (\hat{H} - E) |\psi_{\text{sc}}\rangle &= -(\hat{H}_0 + \hat{V}_{\text{int}} - E) |\psi_0\rangle, \\ (\hat{H} - E) |\psi_{\text{sc}}\rangle &= -\hat{V}_{\text{int}} |\psi_0\rangle. \end{aligned} \quad (1.2)$$

The differential equation (1.2) is the primary equation this thesis aims to solve using the finite element method and iterative numerical methods for the resulting systems of linear equations. Before we introduce them, we will first describe specific models we will study.

We will not go into details of the scattering theory, which can be found e.g. in Taylor[1]. The only utilized quantity is the integral cross-section

$$\sigma_{i \rightarrow f} = \int d\Omega \frac{d\sigma}{d\Omega} = \int d\Omega |f(i \rightarrow f)|^2, \quad (1.3)$$

where $f(i \rightarrow f)$ is the scattering amplitude of the process $i \rightarrow f$. This amplitude can be expressed using the off-shell $T(z)$ matrix as

$$f(i \rightarrow f) = \langle f | S | i \rangle = -(2\pi)^2 m \langle f | T(E_i + i0) | i \rangle, \quad (1.4)$$

where the $+i0$ term stands for a limit $\lim_{\epsilon \rightarrow 0^+} i\epsilon$ and the S matrix relates the input and final states of a scattering experiment. For more details refer to [1] or [2].

Unless explicitly stated otherwise, we will be using the atomic units

$$m_e = \hbar = e = 1. \quad (1.5)$$

1.1 A simple two-channel model

As a first benchmark problem we will consider a simplistic one-dimensional two-channel model, in which the wave function has two components. The Hilbert space is now

$$\mathcal{H} = \mathcal{H}_{\text{sp}} \oplus \mathcal{H}_{\text{sp}}, \quad (1.6)$$

where $\mathcal{H}_{\text{sp}}$ is the Hilbert space of a single particle with one spatial degree of freedom. The states can be represented using a vector function

$$\psi(r) = \begin{pmatrix} \psi_1(r) \\ \psi_2(r) \end{pmatrix} \quad (1.7)$$

where we introduced the coordinate $r \in [0, \infty)$, since we model the radial part of a spherically symmetric problem.

Next we can introduce the potential as a matrix, where the diagonal elements encode the dynamic inside a channel and the off-diagonal elements represent couplings between channels:

$$V(r) = \begin{pmatrix} V_{11}(r) & V_{12}(r) \\ V_{21}(r) & V_{22}(r) \end{pmatrix}. \quad (1.8)$$

Since this is a radial problem, we can also introduce angular momenta l_i for each channel. Assuming $m = 1$, we have the final Hamiltonian

$$\hat{H} = \begin{pmatrix} \hat{T} + \frac{l_1(l_1+1)}{2r^2} + V_{11}(r) & V_{12}(r) \\ V_{21}(r) & \hat{T} + \frac{l_2(l_2+1)}{2r^2} + V_{22}(r) \end{pmatrix}. \quad (1.9)$$

For our purposes we shall use a potential which is piecewise constant, defined for a certain distance r_0 and constant real symmetric matrix v_{ij} as

$$V_{ij}(r) = \begin{cases} E_i \delta_{ij} & r \geq r_0, \\ v_{ij} + E_i \delta_{ij} & r < r_0, \end{cases} \quad (1.10)$$

where we additionally explicitly introduced channel energy levels E_i . The initial state is a free wave in one of the (open) channels. For given input energy E we can calculate the momentum in each channel $k_i = \sqrt{2(E - E_i)}$ and the initial state is then

$$\psi_{0,1}(r) = \sqrt{\frac{2}{\pi k_1}} \hat{j}_{l_1}(k_1 r) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{or} \quad \psi_{0,2}(r) = \sqrt{\frac{2}{\pi k_2}} \hat{j}_{l_2}(k_2 r) \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad (1.11)$$

where $\hat{j}_i(z) = z j_i(z)$, $\hat{n}_i(z) = -r n_i(z)$ are the Riccati-Bessel and Riccati-Neumann functions and where j_i and n_i are the spherical Bessel functions of the first and the second kind[3]. We now have all ingredients to solve the equation (1.2). The cross-section is given by

$$\sigma_{i \rightarrow j} = \frac{4\pi^3}{k_i^2} |\langle \psi_{0,j} | V | \psi_i \rangle|^2, \quad (1.12)$$

where $|\psi_i\rangle = |\psi_{0,i}\rangle + |\psi_{\text{sc}}\rangle$ obtained from (1.2).

This simple case can be solved in the closed form as shown in [2]. The cross-section can be expressed via the following quantities

$$\begin{aligned}
R_i &= k_i^2 - 2v_{ii}r_0^2 \\
\epsilon_i &= \frac{1}{2}\sqrt{R_1 + R_2 - (-1)^i [(R_1 - R_2)^2 + 16v_{12}^2r_0^4]} \\
b_i &= \epsilon_i - R_1 \\
c_i &= \frac{\sqrt{\epsilon_i}}{\tan \sqrt{\epsilon_i}} \\
G(x, y) &= (b_1 - b_2)(xy - c_1c_2) + i[x(c_1b_1 - c_2b_2) + y(c_2b_1 - c_1b_2)]
\end{aligned} \tag{1.13}$$

as

$$\begin{aligned}
\sigma_{j \rightarrow j} &= 1 - e^{-2ik_j} \frac{G((-1)^j k_1, (-1)^{j+1} k_2)}{G(k_1, k_2)} \\
\sigma_{1 \rightarrow 2} &= 2(c_1 - c_2)e^{-i(k_1+k_2)} \frac{\sqrt{b_1 b_2 k_1 k_2}}{G(k_1, k_2)}
\end{aligned} \tag{1.14}$$

For our calculations, we chose the following constants:

$$r_0 = 1, \quad v_{11} = v_{22} = -2, \quad v_{12} = v_{21} = -\frac{1}{2}, \quad E_1 = 0, \quad E_2 = 2 \tag{1.15}$$

and no angular momenta ($l_i = 0$). The analytic solution (1.14) can be seen in the graph 1.1. A resonance can be observed in the cross-section $\sigma_{1 \rightarrow 1}$ at $E \approx 1.85$.

1.2 Two-dimensional diatomic models

These models were introduced in [4]. They describe the interaction between an incoming electron and a diatomic molecule and are used to study vibrational excitation

$$e^- + AB(\nu_i) \rightarrow e^- + AB(\nu_f) \tag{1.16}$$

and dissociative electron attachment

$$e^- + AB(\nu_i) \rightarrow A + B^-. \tag{1.17}$$

The significant difference from the previous model is that instead of two channels we are now dealing with two spatial dimensions, namely the electron distance r and diatomic distance R . This circumstance makes this model much more difficult to solve.

This model is again described by a Hamiltonian $\hat{H} = \hat{H}_0 + \hat{V}_{\text{int}}(r, R)$, where $\hat{V}_{\text{int}}$ is the interaction potential and the free Hamiltonian is now of the form

$$\hat{H}_0 = \hat{T}_R + \hat{V}_0(R) + \hat{T}_r + \frac{l(l+1)}{2r^2}, \quad \hat{T}_R = -\frac{1}{2\mu} \frac{d^2}{dR^2}, \quad \hat{T}_r = -\frac{1}{2} \frac{d^2}{dr^2}, \tag{1.18}$$

where $V_0(R)$ denotes the inter-atomic potential of the diatomic molecule.

The incoming electron with momentum k and angular momentum l is in the energy-normalized continuum state described via the spherical Bessel functions as

$$\mathcal{J}_k^l(r) = \sqrt{\frac{2k}{\pi}} r j_l(kr). \tag{1.19}$$

Meanwhile, the initial state of the molecule is its chosen vibrational state identified by its index ν_i and described by wave function χ_{ν_i} , which is an eigenstate of the Hamiltonian $H_R = T_R + V_0(R)$ with energy E_{ν_i} . The complete initial state is given as direct product of these two states

$$|\psi_0\rangle = |\chi_{\nu_i} \mathcal{J}_{k_i}^l\rangle \quad (1.20)$$

and has total energy

$$E = E_{\nu_i} + \frac{k_i^2}{2}. \quad (1.21)$$

The vibrational excitation T matrix for this problem is given by

$$T_{v_i \rightarrow v_f}^{\text{VE}}(E) = \langle \chi_{\nu_f} \mathcal{J}_{k_f}^l | \hat{V}_{\text{int}} | \psi \rangle, \quad (1.22)$$

where ν_f is a chosen final vibrational state, $k_f = \sqrt{2(E - E_{\nu_f})}$ is the electron's final momentum and $|\psi\rangle = |\psi_0\rangle + |\psi_{\text{sc}}\rangle$ is again the solution of the scattering problem obtained by solving (1.2).

The dissociative attachment T matrix is given by

$$T_{v_i}^{\text{DA}}(E) = \langle \mathcal{E}_K(R) \phi_b | \hat{V}_{\text{int}} + \hat{V}_0 - \hat{V}_b | \psi \rangle, \quad (1.23)$$

where

$$\mathcal{E}_K(R) = \sqrt{\frac{2\mu}{\pi K}} \sin(KR) \quad (1.24)$$

is the energy-normalized molecular continuum state (relative outward motion of atoms) with momentum K , where μ is the reduced mass of the molecule. Next

$$V_b(r) = \lim_{R \rightarrow \infty} V_{\text{int}}(R, r) \quad (1.25)$$

is the interaction of electron and the diatomic molecule for large internuclear distances and ϕ_b is the electronic bound state corresponding to the B^- atom, an eigenstate of $H_r = T_r + l(l+1)/2r^2 + V_b(r)$ with eigenvalue $E_b = -E_a$ where E_a is called the electron affinity and corresponds to the electron affinity of atom B .

We can then calculate the cross sections for these two processes

$$\sigma_{\nu_i \rightarrow \nu_f}^{\text{VE}}(E) = \frac{4\pi^3}{k_i^2} \left| T_{v_i \rightarrow v_f}^{\text{VE}}(E) \right|^2 \quad (1.26)$$

$$\sigma_{\nu_i}^{\text{DA}}(E) = \frac{4\pi^3}{k_i^2} \left| T_{v_i}^{\text{DA}}(E) \right|^2 \quad (1.27)$$

The molecules we are interested in are N_2 , NO , F_2 and O_2 . The first three molecules were already solved and discussed in [4]. The model for oxygen is introduced here and requires a different treatment. The interaction is given by

$$V_{\text{int}}(r, R) = -\lambda(R) e^{-\alpha_{\text{O}} r^2}. \quad (1.28)$$

For the first three molecules the Morse potential is used for $V_0(R)$, and the following parametrization is utilized:

$$\begin{aligned} V_0(R) &= D_0 \left(e^{-2\alpha_0(R-R_0)} - 2e^{-\alpha_0(R-R_0)} \right), \\ \lambda(R) &= \lambda_\infty + (\lambda_C - \lambda_\infty) \frac{1 + e^{\lambda_1(R_C - R_\lambda)}}{1 + e^{\lambda_1(R - R_\lambda)}}. \end{aligned} \quad (1.29)$$

The specific values are shown in the table 1.1. The total effective potentials are plotted in the graph 1.2.

	N ₂	NO	F ₂
μ	12766.36	13614.16	17315.99
l	2	1	1
D_0	0.75102	0.2363	0.0598
α_0	1.15350	1.5710	1.5161
R_0	2.01943	2.1570	2.6906
λ_∞	6.21066	6.3670	18.8490
λ_C	5.38022	6.0500	18.1450
λ_1	1.05708	5.0000	3.2130
R_C	2.40500	2.2850	2.5950
R_λ	-27.9833	2.0843	1.8320
α_C	0.4	1.0	3.0

Table 1.1 Model parameters for different diatomic molecules

The O₂ molecule

The O₂ molecule requires special treatment, because its *ab initio* potential energy curves [5] are not well approximated by the Morse potential and the two-dimensional model for this molecule has not been published yet. For this molecule, the following, more elaborated parametrization of the potential $V_0(R)$ was constructed [6] which approximates reasonably well the lowest-lying electronic state of O₂:

$$\begin{aligned}
 V_0(R) &= \begin{cases} 4.4 & R < 0.6, \\ -V_A(R) - V_B(R) & R \geq 0.6, \end{cases} \\
 V_A(R) &= \frac{D_0}{2} e^{-\alpha_0(R-R_0)} p_1(R - R_0) \left\{ 1 - \tanh \left[c_1 (p_2(R) e^{c_2 p_2(R)} - 1) \right] \right\}, \\
 p_1(z) &= 1 + \alpha_0 z - 0.197253 z^2 - 0.453587 z^3, \\
 p_2(z) &= z - 4.25451, \quad c_1 = 0.589844, \quad c_2 = 3.19454, \\
 V_B(R) &= \frac{c_3 R^2 - c_4}{2R^6} \left\{ 1 + \tanh \left[c_5 (p_3(R) + 1 - e^{-c_6 p_3(R)}) \right] \right\}, \\
 p_3(z) &= z - 2.63224, \\
 c_3 &= 1.263645, \quad c_4 = 28.9218, \quad c_5 = 0.270386, \quad c_6 = 2.55552.
 \end{aligned} \tag{1.30}$$

To obtain a good correspondence of the 2D model to the dynamics on the lowest-lying electronic state of the negative molecular ion O₂⁻, the $\lambda(R)$ portion of interaction potential now has an extra term and is given by

$$\lambda(R) = \lambda_\infty + \frac{\lambda_1}{1 + e^{\alpha_1(R-R_1)}} + \frac{\lambda_2}{1 + e^{\alpha_2(R-R_2)}}. \tag{1.31}$$

The constants are shown in the table 1.2

	O ₂
μ	14682.6
l	2
D_0	0.23178
α_0	1.96203
R_0	2.29989
λ_∞	1.97920
λ_1	-38.9401
α_1	1.63986
R_1	-0.448295
λ_2	0.230156
α_2	1.480600
R_2	4.464780
α_C	0.133

Table 1.2 Model parameters for O₂

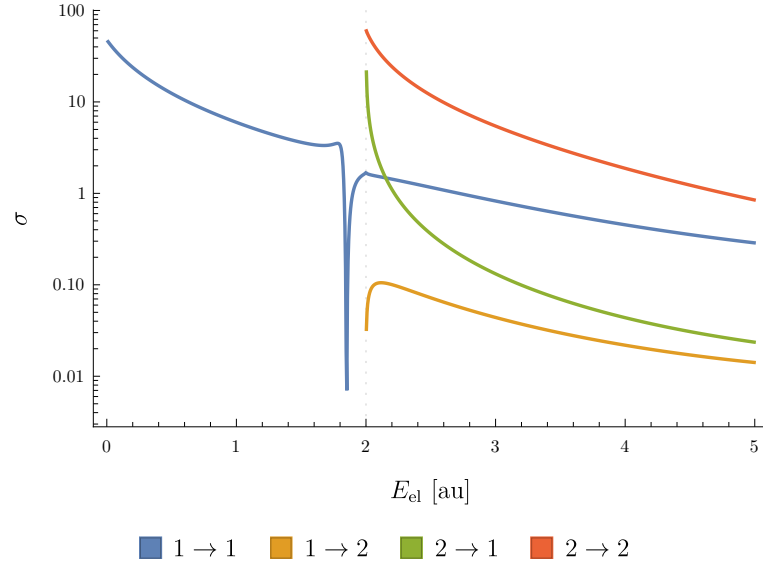


Figure 1.1 The cross-sections from the analytic solution of the two-channel model.

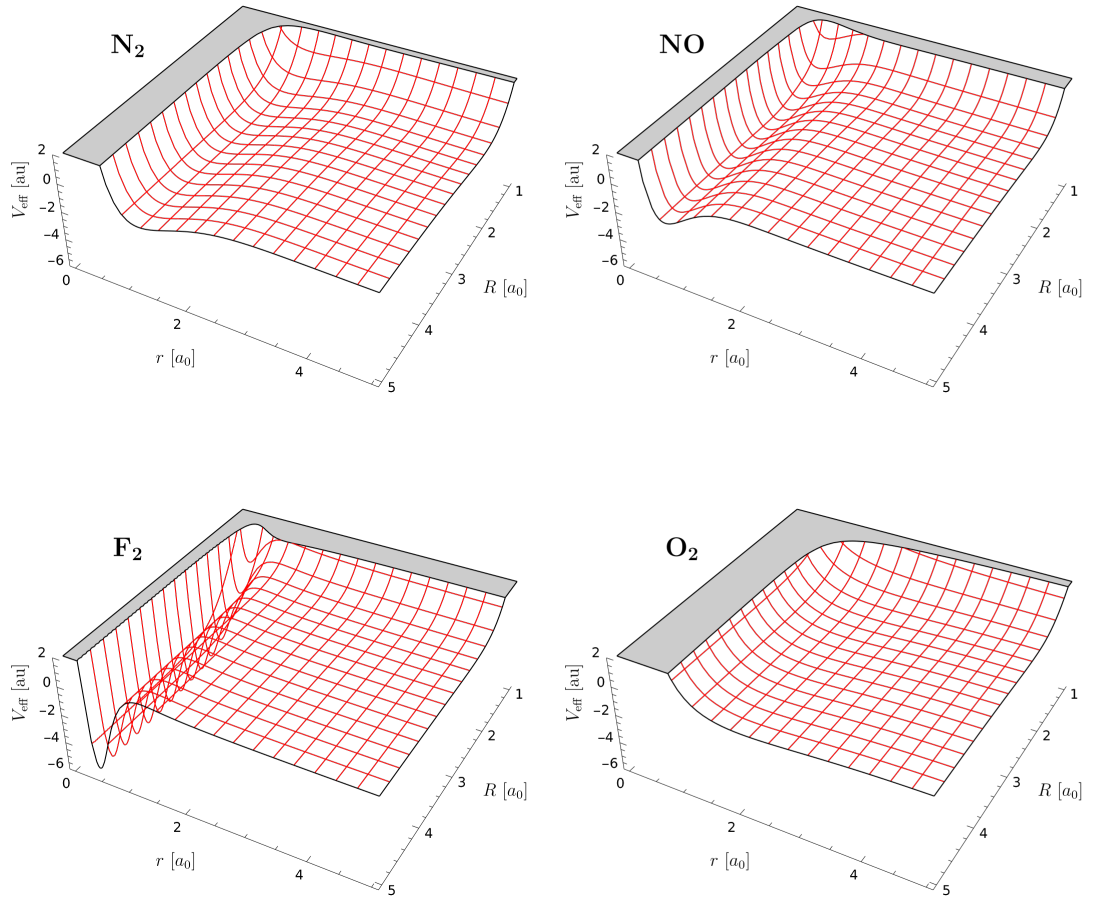


Figure 1.2 The effective potential $V_{\text{eff}}(R, r) = V_0(R) + V_{\text{int}}(R, r) + \frac{l(l+1)}{2r^2}$ for all used molecular parameters.

Chapter 2

Mathematical background

In this chapter we are going to introduce the necessary tools to represent and solve the differential equations of the type:

$$(E - \hat{H}) |\psi_{sc}\rangle = \hat{V}_{int} |\psi_0\rangle \quad (2.1)$$

derived from the Schrödinger equation in the previous chapter.

2.1 Finite element method

This chapter is roughly inspired by Rescigno and McCurdy[7]. The finite element method is an alternative to the finite difference method, in which functional derivatives are replaced by differences of function values in a certain neighborhood of points of a discretized interval. Consider a partial differential equation of order d

$$R(\psi, \partial\psi, \partial^2\psi, \dots, \partial^d\psi) = \partial^d\psi + \bar{R}(\psi, \dots, \partial^{d-1}\psi) = f(x), \quad (2.2)$$

where $\psi(x)$ is a function with arbitrary tensorial character, x is a vector of some dimension D and $\partial^i\psi$ is some i -th derivative of this function. We call this the *strong formulation* of a problem. We can also introduce the *weak formulation*. By multiplying both sides of the equation by an arbitrary weight function $w(x)$ and integrating over Ω we get:

$$\int_{\Omega} dx \, R w = \int_{\Omega} dx \, f w \quad \forall w(x). \quad (2.3)$$

Now we can integrate the term with the highest order of derivative $\partial^d\psi$ per partes to obtain the form

$$\int_{\Omega} dx \, [\bar{R}(\psi, \dots, \partial^{d-1}\psi) w - \partial^{d-1}\psi \, w'] = \int_{\Omega} dx \, f w \quad \forall w(x), \quad (2.4)$$

where the boundary term disappears because the test functions must all vanish on the boundary. This is precisely the weak formulation of problem (2.2). We can also neatly rewrite it into a much less syntax dense form by introducing the scalar product $(f, g)_{\Omega} = \int_{\Omega} dx \, f g$:

$$(\partial^{d-1}\psi, w')_{\Omega} = (\bar{R} - f, w)_{\Omega} \quad \forall w. \quad (2.5)$$

This formulation allows us to look for the solution using a certain space of functions on Ω which we will denote V . Further simplification can be obtained by restricting ourselves to a subset of V , for example generated by a finite number of basis functions y_i (any function can be expanded to $f(x) \approx \sum c_i y_i(x)$, where c_i are constant). This is generally called the *Galerkin method*. If we choose the space V as the space of continuous piecewise polynomial functions, we finally obtain the *finite element method*. We will skip the formalities of general FEM and will instead jump directly to a specific choice of basis in a method called the *discrete variable representation*.

Before we continue, a few general notes are worth a mention. Generally in FEM we choose a finite partitioning of Ω , which defines the piecewise structure of the basis. This step is called *discretization*. Weak formulation in combination with certain basis functions permits discontinuities in derivatives. This is actually a convenient feature relevant in most engineering fields, because it allows for a better simulation behavior near surfaces and material transitions, when discretized correctly.

2.1.1 Discrete variable representation

To illustrate this method, we will limit ourselves to a one-dimension case with coordinate $r \in \mathbb{R}_0^+$ (we will generalize it to the multidimensional case using a correct product). Next, we partition a chosen finite sub-interval on which r exists by choosing a finite number of distinct points $\{R_i\}_{i=0}^N$ called *nodes*. In the 1D case in $\mathbb{R}_0^+$ it's valid to consider these points ordered with $R_0 = 0$. The intervals (R_i, R_{i+1}) are then called elements. On each element we can then define a polynomial basis.

For DVR, the basis is constructed in the following way. First, on each element we can integrate polynomials up to a certain order exactly using Gaussian quadratures. A quadrature rule of particular interest for this method is the Gauss-Lobatto quadrature, because it evaluates the polynomial in the endpoints of an interval, which is useful because of bridging. A Gauss-Lobatto rule of order n evaluates the function at n points and is exact for polynomials up to order $2n - 3$. An integral on the interval $[-1, 1]$ then becomes

$$\int_{-1}^1 dx f(x) \approx \sum_{j=1}^n w_j^{\text{GL}} f(x_j^{\text{GL}}). \quad (2.6)$$

Every interval $[R_i, R_{i+1}]$ can be mapped onto from the interval $[-1, 1]$ using the linear map $r_i(x) = R_i + \frac{x+1}{2}(R_{i+1} - R_i)$. Using this map, we can map the Gauss-Lobatto quadrature points from $[-1, 1]$ onto each element. Since $dr_i = \frac{R_{i+1}-R_i}{2}dx$, we immediately see that weights have to be scaled by half of the element's length

$$w_{i,j} = \frac{R_{i+1} - R_i}{2} w_j^{\text{GL}}. \quad (2.7)$$

Now we have every element starting at R_i equipped with a set of quadrature points $\{r_{i,j}\}_{j=1}^n$ and weights $\{w_{i,j}\}_{j=1}^n$. It is also automatically true that $r_{i,n} = r_{i+1,1}$.

Next we define a polynomial basis on each element. The key feature of DVR is using Lagrange polynomials as the basis functions, which are defined as zero

outside the element they belong to. For each point in the element starting at R_i , we have a polynomial

$$\tilde{L}_{i,j}(r) = \begin{cases} \frac{1}{\sqrt{w_{i,j}}} \prod_{k \neq j}^n \frac{(r-r_{i,k})}{r_{i,j}-r_{i,k}} & r \in [R_i, R_{i+1}] \\ 0 & r \notin [R_i, R_{i+1}], \end{cases} \quad (2.8)$$

where the normalization factor is significant, because

$$(\tilde{L}_{i,j}, \tilde{L}_{i,k})_{[R_i, R_{i+1}]} \approx \sum_{l=1}^n \frac{w_{i,l}}{\sqrt{w_{i,j}w_{i,k}}} \tilde{L}_{i,j}(r_{i,l}) \tilde{L}_{i,k}(r_{i,l}) = \sum_{l=1}^n \frac{w_{i,l}}{\sqrt{w_{i,j}w_{i,k}}} \delta_{jl} \delta_{kl} = \delta_{jk},$$

and the basis is thus orthonormal with respect to the integration by the quadrature rule (this distinction is important, because in the scalar product we're integrating a polynomial of order $2n-2$ and our quadrature only guarantees exactness for polynomials of order $2n-3$, which is never sufficient).

To be able to use this basis globally, the last thing we have to do is to resolve the conflict on the nodal points. The respective polynomials from the neighboring elements are equal to their respective normalization factor at this point and their supports intersect only here. Therefore, we can combine these two functions by normalizing them to the same value and then by addition to obtain a new function which bridges the two elements and (almost) does not violate the orthonormality of the basis (globally). Although it is orthogonal to every other basis function, the function is not normalized. To normalize it, we have to realize that this point now carries the quadrature weight from both of the elements, and we have to scale it as

$$L_{i,1}(r) = \frac{\sqrt{w_{i-1,n}} \tilde{L}_{i-1,n}(r) + \sqrt{w_{i,1}} \tilde{L}_{i,1}(r)}{\sqrt{w_{i-1,n} + w_{i,1}}}. \quad (2.9)$$

To clean the notation up, we are going to move from the per-element basis setup to a global basis setup by dropping every endpoint of each element except for the last one $r_{i \neq (N-1),n}$ while introducing the numbering $r_{i,j} \rightarrow r_{in+j}$ and using the bridged polynomial for each initial point except for the first element. This way we end up with $(n-1)N+1$ points $\{x_i\}$, weights $\{w_i\}$ and basis functions $\{L_i\}$.

Before we can start using this basis, we have to determine the matrices of relevant quantum operators. First, let's investigate potentials $V(r)$:

$$V_{ij} = (L_i, V L_j)_\Omega \approx \sum_k w_k V(r_k) L_i(r_k) L_j(r_k) = \sum_k \frac{w_k}{\sqrt{w_i w_j}} \delta_{ik} \delta_{jk} V(r_k) = \delta_{ij} V(r_i). \quad (2.10)$$

The potential is diagonal, and its diagonal elements are the functional values at the associated quadrature points. This is the reason this representation is called DVR.

The second relevant operator is the kinetic operator:

$$-2mT_{ij} = - \int_\Omega L'_i L'_j \approx - \sum_k w_k L'_i(r_k) L'_j(r_k), \quad (2.11)$$

The remaining terms have to be evaluated and together create what is called the *stiffness matrix*.

2.1.2 Exterior complex scaling

The previous remark stated that the boundary condition will require the wave function to be zero on both endpoints. Since we work on a compact subset of $\mathbb{R}_0^+$, this is however impossible to ensure consistently, because the scattered wave function is asymptotically harmonic. The oscillatory behavior is of the form $f(r) \sim e^{ikr}$. If we use a complex number $r = a + ib$, the behavior now becomes

$$\psi_{\text{sc}}(a + ib) \sim e^{-kb} e^{ika} \quad (2.12)$$

and using the parameter b we can dampen the wave function to be arbitrarily low.

This is the core idea behind exterior complex scaling (ECS) method which will allow us to enforce the desired boundary condition. To start off, we will choose a point R_{ECS} inside the subset covered by our elements from which we redirect our axis into the complex plane under an angle θ . This can be stated formally using the coordinate transformation:

$$R(r) = \begin{cases} r & r \leq R_{\text{ECS}}, \\ R_{\text{ECS}} + (r - R_{\text{ECS}})e^{i\theta} & r > R_{\text{ECS}}. \end{cases} \quad (2.13)$$

To make this transformation compatible with our choice of points we ensure R_{ECS} is exactly one of the nodes R_i , which we will denote R_I . The illustrated way of scaling weights (2.7) still applies, and it is not necessary to perform any more adjustments, only now the weights are complex.

One thing to note is that the chosen linear map is continuous, but it is not smooth. In fact, there is a discontinuity in the first derivative. This will explicitly create discontinuities in the derivatives of solutions of Schrödinger equation, but this is a setting FEM is designed for, so no numerical issues are to be expected. Although it is possible to construct a smooth ECS mapping, it would provide no benefit in our case and would only make the implementation cumbersome.

To summarize, here is how we use the FEM-DVR-ECS method:

1. Discretize a finite subset of the problem's domain using $N + 1$ nodes R_i , obtaining N elements.
2. Pick one node R_I sufficiently distant from the interaction region for ECS with angle θ .
3. Choose the order of Gauss-Lobatto quadrature n and use its points and weights to generate $(n - 1)N + 1$ grid points r_i each with its weight w_i .
4. Represent the equations using operator matrices and vectors using the basis made of Lagrange polynomials.
5. Solve the system of linear equations $Ax = b$ in this representation. For the equation (1.2), the left-hand side matrix is $A_{ij} = T_{ij} + V_{ij} - E\delta_{ij}$ and the right-hand side vector is $b_i = -V_{\text{int},ii}\psi_{0,i}$.

2.2 Solving linear systems

The finite element method transforms the differential equation into a system of linear equations. For good approximations, these systems are quite large and efficient algorithms are needed to solve them in feasible time with the available amount of memory. Generally, we are solving systems of type

$$Ax = b, \tag{2.14}$$

where $x, b \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n \times n}$. There are two types of algorithms for linear systems, direct and iterative. The direct algorithms generally compute some factorization of the matrix A or use a certain formula with a given number of steps to obtain the exact solution. Iterative methods use the original matrix A and repeatedly apply it to obtain an approximate solution where the amount of steps required for a certain precision is unknown a priori.

Direct methods are preferred when working with linear systems such as those we have introduced in the FEM-DVR-ECS section because of their predictability. Iterative methods can take too long to achieve a desired precision or not even converge at all. The killer feature of iterative methods is the fact they utilize the original matrix. Many problems (including the ones we are interested in) involve matrices A with very little non-zero elements compared to the total size of the matrix. These types of matrices are called *sparse* and can be represented in memory far more efficiently. Factorizations utilized in direct methods generally do not produce sparse decompositions for sparse matrices and the memory requirements can become large. Iterative methods can be further improved via *preconditioning*.

2.2.1 Direct methods

The general problem (2.14) can be solved directly using the inverse matrix $x = A^{-1}b$. Although it can be directly calculated for example with the Cramer's rule (complexity $O(n! \cdot n)$) or Gaussian elimination ($O(n^3)$), the computation takes a significant amount of time, and it is cheaper to compute some factorization of A for which the inverse action on a vector b can be performed without calculating an additional matrix. Three such types of matrices are the diagonal matrices, for which the inverse action is simply $x_i = b_i/A_{ii}$ and the lower and upper triangular matrices, for which the forward and backward substitutions can be used.

Computing the factorization is not needed for problems where only one right-hand side is ever used. There, the Gaussian elimination with complexity $O(n^2)$ or many more efficient algorithms that respect the symmetries and attributes of the matrix are available. The details are beyond the scope of this thesis and are not relevant, as they have been the centerpiece of previous work in this field. We will only be using direct methods for comparison, namely the *zsysv* method from LAPACK[8] for the two-channel model and the Intel Direct Sparse Solver[9] for the two-dimensional case.

2.2.2 Iterative methods

The iterative methods rely on repeated application of some operation F derived from the problem matrix A and the right-hand side b

$$x^{k+1} = F(x^k), \quad r_e^k = x^k - x^{\text{exact}}, \quad Ax^{\text{exact}} = b, \quad (2.15)$$

such that $|r_e^k|$ tends to zero. We also define the residual vector

$$r^k = b - Ax^k, \quad (2.16)$$

whose norm $\|r^k\|$ (which we refer to as the *residuum*) is a useful quantity to use as a halting condition. There are two types of these methods, the *stationary* methods and methods based on *Krylov subspaces*, which are utilized in this work. A Krylov subspace for the system $Ax = b$ is the linear subspace

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, \dots, A^k b\}. \quad (2.17)$$

There are many methods utilizing these subspaces to approximate the solution of the problem. For real symmetric positive definite matrices the basic method is the conjugate gradient (CG) method. To lift the positive definite criterion, the more computationally costly conjugate residual (CR) method can be constructed. This method works for all Hermitian matrices. However, our case is complex symmetric matrices. We will be using a modified CR method called the conjugate orthogonal conjugate residual (COCR) method[10]. There, the series of approximates is obtained via the relations:

$$\begin{aligned} x^{k+1} &= x^k + \alpha^k p^k, & p^k &= r^k + \beta^{k-1} p^{k-1}, & p^{-1} &= 0, \\ \alpha^k &= \frac{(Ar^k, r^k)}{(Ap^k, Ap^k)}, & \beta^{k-1} &= \frac{(Ar^k, r^k)}{(Ar^{k-1}, r^{k-1})}, & \beta^{-1} &= 0, \end{aligned} \quad (2.18)$$

where x_0 is an arbitrary initial guess and $(\cdot, \cdot)$ is a bilinear (not sesquilinear) dot product. In the cited paper the product is sesquilinear, so the first parameter is conjugated in each term. For more details refer to [10].

As a condition for stopping the simulation, we compare the quantity

$$\xi^k = \frac{\|r^k\|}{\|b\|} \quad (2.19)$$

with a given constant (for the most part with $\xi_t = 10^{-8}$). The halting condition for the iterative method is then $\xi^k \leq \xi_t$. We refer to ξ^k (used also without the superscript k) as the *RHS-relative residuum*.

2.2.3 Preconditioning

The convergence of the presented iterative methods can be very slow. The condition number

$$\kappa(A) = \frac{|\lambda|_{\max}}{|\lambda|_{\min}} \geq 1, \quad (2.20)$$

where $|\lambda|_{\max|\min}$ is the maximal and minimal absolute eigenvalue respectively, is an important metric of how well a given matrix problem converges. Typically,

the lower the condition number, the faster iterative methods converge, although this is not guaranteed.

A common technique for lowering the condition number is known as preconditioning. Commonly we compute a cheap factorization-like matrix with similar sparsity pattern which approximates the spectrum of A well. One such factorization we will use is the incomplete LU decomposition with zero fill-in (ILU0). It is a factorization of the form

$$A = LU + R, \quad (2.21)$$

where L and U are lower and upper triangular respectively and also zero outside the sparsity pattern of A , R is zero in the sparsity pattern and technically not computed, as only the L and U matrices are used. Zero fill-in refers to the fact that no element outside the sparsity pattern is added in this decomposition.

To utilize the preconditioned matrix efficiently, we will use a modified version of the COCR algorithm described in [11].

Partial inverse

In the two-dimensional model, the stiffness matrix is composed of block diagonal and off-block diagonal strips. We introduce a special preconditioner which inverts the diagonal blocks directly and ignores all off-block diagonal elements. This preconditioner is referred to as *partial inverse* in this work. By definition, this preconditioner does not preserve the sparsity pattern of the original matrix, by both omitting the off-block diagonal elements and by destroying the sparsity of one-dimensional problem stiffness matrix blocks by inverting. This negatively impacts performance, because the number of multiplications per iteration rises. The effect of matrix inversion on the matrix density for a general stiffness matrix is shown in the graph 2.1.

2.2.4 Kickstart

When initializing an iterative solver, the initial guess x_0 can be chosen arbitrarily. Generally our choice of the initial guess is $x_0 = 0$. The idea behind a kick-started method is to use the solution of a run close in the parametric space as the initial guess. In theory, this could skip a number of steps at the beginning of the run. Its effectiveness is benchmarked in the next chapter.

2.3 Sparsity pattern of the left-hand side

Before we show the calculated results and benchmarks, we shall discuss the sparsity of the left-hand side matrix A for our models. Suppose we have a Gauss-Lobatto quadrature of order G . For a one-dimensional grid with N elements, we have a stiffness matrix with N full blocks of size $G \times G$ where two neighboring elements share the boundary point and thus the blocks overlap at this point. This means we have a number of non-zero elements

$$S_T = N(G^2 - 1) + 1. \quad (2.22)$$

The number of non-zero elements is linear in the number of elements N and quadratic in the quadrature order G . A density plot of an example with Gauss-Lobatto order $G = 5$ and $N = 3$ elements is shown in the graph 2.2.

2.3.1 The two-channel Hamiltonian

The two-channel Hamiltonian (1.9) is constructed by inserting two stiffness matrices as diagonal blocks and then putting the coupling potential diagonal submatrices into the two off-diagonal block slots. This results in a number of non-zero elements

$$S = 2S_T + 2[N(G - 1) + 1] = 2N(G - 1)(G + 2) + 4. \quad (2.23)$$

The dependence on N and G is of the same complexity as for the one-dimensional stiffness matrix. A density plot of this pattern is shown in the graph 2.3 for $G = 5$ and $N = 3$.

2.3.2 The two-dimensional Hamiltonian

For the multidimensional case the dependence is much more complicated. First, assume we have two distinct grids covering subsets Ω_1 and Ω_2 for coordinates x and y with number of elements N_1 and N_2 and Gauss-Lobatto orders G_1 and G_2 . We construct the two-dimensional grid as a Cartesian product of these two grids. This results in $n = n_1 n_2 = [N_1(G_1 - 1) + 1][N_2(G_2 - 1) + 1]$ points. First observe that the integration on this two-dimensional surface is of the form

$$\int_{\Omega} dx dy f(x, y) = \int_{\Omega_1} dx \int_{\Omega_2} dy f(x, y) \approx \sum_{i_1} \sum_{i_2} w_{i_1}^1 w_{i_2}^2 f(x_{i_1}^1, x_{i_2}^2), \quad (2.24)$$

where $\{x^j\}$ and $\{w^j\}$ are points and weights for each independent grid. We can now rewrite this using n tuples $\{(x_i, y_i)\}_{i=1}^n$ and weights $\{w_i\}_{i=1}^n$, where the tuples are simply the points obtained from the Cartesian product and the weights are products of the weights at the corresponding independent grid points. The only ambiguity here is the ordering of the product points as there is $n!$ ways of rearranging them.

We will consider only two specific orderings which we will denote 12 and 21 (or "me" and "em" in the results, standing for "molecular" and "electronic" grid). Ordering 12 is constructed by taking the first point $y_1 \in \Omega_2$ and sequentially iterating through $x_i \in \Omega_1$, then incrementing to $y_2 \in \Omega_2$ and iterating sequentially through $x_i \in \Omega_1$ again, until exhausting all $y_i \in \Omega_2$. Ordering 21 is the same principle but with swapped axes, first we fix $x_1 \in \Omega_1$ and iterate through $y_i \in \Omega_2$ sequentially, etc.

With an ordering selected, we can evaluate the integral as

$$\int_{\Omega} dx dy f(x, y) \approx \sum_{i=1}^n w_i f(x_i, y_i). \quad (2.25)$$

The Lagrange polynomials are also simply constructed using products, the weights are automatically correct this way. The same principle is straightforwardly generalized to an arbitrary dimension D , but this will not be explored in this work any further.

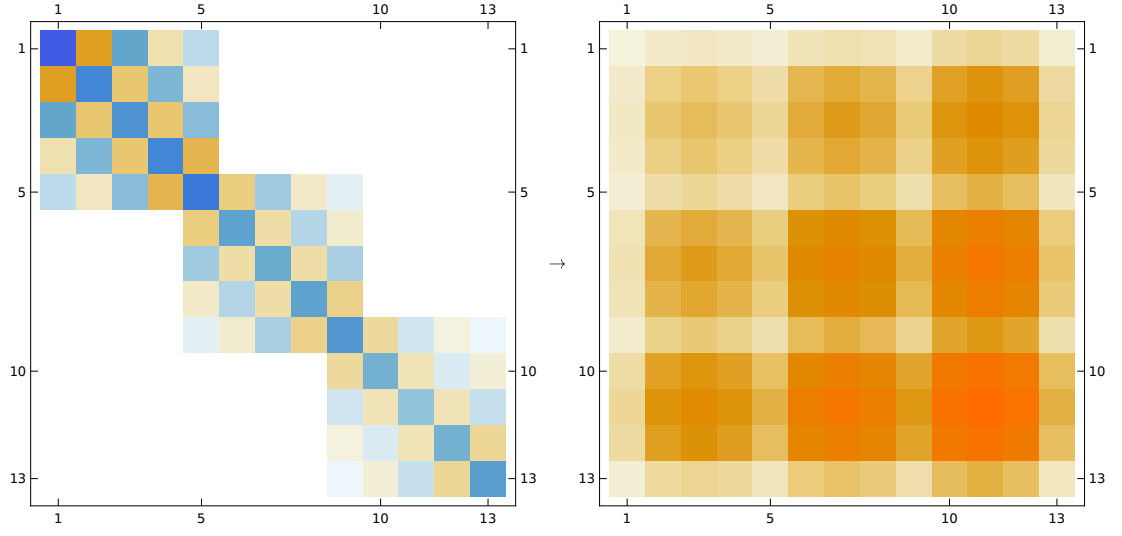


Figure 2.1 The sparsity pattern of a one-dimensional stiffness matrix T for $G = 5$ and $N = 3$ (with elements of lengths 1, 2 and 3) and its inverse T^{-1} .

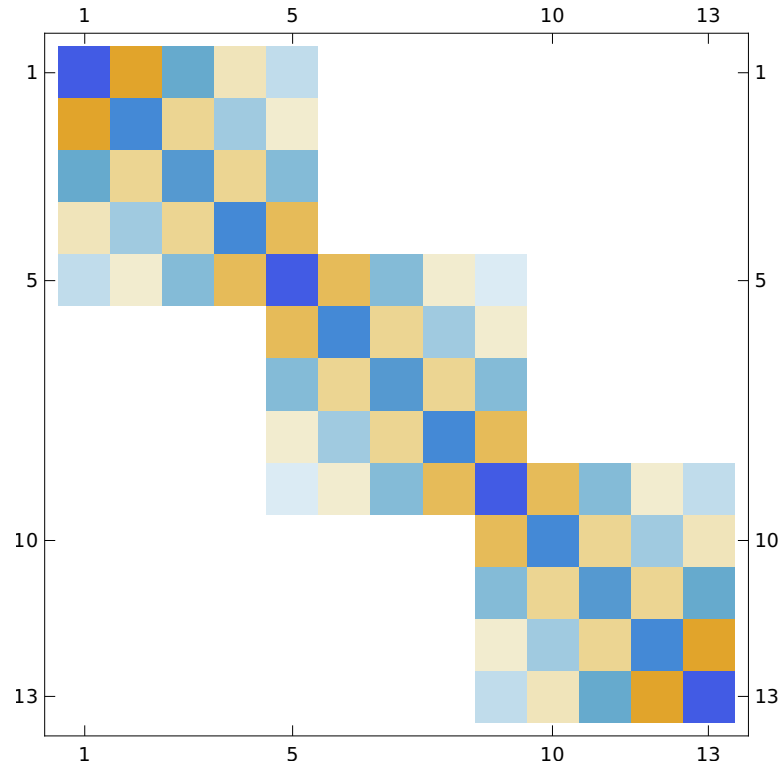


Figure 2.2 The sparsity pattern of a one-dimensional stiffness matrix T for $G = 5$ and $N = 3$.

The total stiffness matrix is the sum of the stiffness matrices for each dimension $T = T_1 \otimes \mathbb{1}_2 + \mathbb{1}_1 \otimes T_2$. In the product basis (which is still orthonormal with respect to the Gauss-Lobatto quadrature), the elements of the stiffness matrix have the form

$$T_{ij} = T_{1,ij} \delta(y_i - y_j) + T_{2,ij} \delta(x_i - x_j), \quad (2.26)$$

where the delta matrices ensure that the product grid points are equal in the respective coordinate. This means the number of non-zero elements of the matrix is equal to

$$\begin{aligned} S &= S_{T_1} n_2 + S_{T_2} n_1 - n = \\ &= N_1 N_2 (G_1 - 1)(G_2 - 1)(G_1 + G_2 + 1) + N_1(G_1^2 - 1) + N_2(G_2^2 - 1) + 1 \\ &= N_1 N_2 (G_1 - 1)(G_2 - 1)(G_1 + G_2 + 1) + S_{T_1} + S_{T_2} - 1. \end{aligned} \quad (2.27)$$

This has the same quadratic dependence in G_1 and G_2 and linear in N_1 and N_2 as for the one-dimensional case. A new behavior emerges, when we make both independent grids share parameters $N_1 = N_2 = N$ and $G_1 = G_2 = G$. Now, the number of non-zero elements is quadratic in N and cubic in G . This means the two-dimensional case is much more demanding when refining the grid in both dimensions simultaneously.

A full matrix representation has the number of elements

$$F = n^2 = [N_1(G_1 - 1) + 1]^2 [N_2(G_2 - 1) + 1]^2, \quad (2.28)$$

which results in a quartic dependence on both shared N and G . To obtain accurate results, Gauss-Lobatto quadratures of orders at least $G = 15$ with at least $N = 16$ elements are commonly used. This results in a number of elements $S = 1\,462\,273$ for the sparse matrix case and $F = 2\,562\,890\,625$ for the full matrix case. If these matrices were represented using 128-bit complex numbers, the full matrix would require 38.19 GiB of memory, whereas the sparse case (assuming a suboptimal coordinate format indexed by 32-bit i and j coordinates) would require only 32.68 MiB, which is 0.086 % of the full matrix case.

For a case of $G_1 = G_2 = 5$, $N_1 = 3$ and $N_2 = 4$, a density plot of the stiffness matrix is shown in the graph 2.4 for both orderings 12 and 21. The stiffness matrix of the coordinate, which is iterated through first, is seen repeated in blocks of length of side equal to $N_1(G_1 - 1) + 1 = 13$ on the diagonal. It is repeated exactly $N_2(G_2 - 1) + 1 = 17$ times, resulting in the total matrix with length of side equal to $13 \cdot 17 = 221$. The structure of the stiffness matrix of the second coordinate can be recognized in the strips around this block diagonal structure. In the case of 12 ordering, there are 4 major blocks of these stripes connecting at the neighboring points. Since the quadrature in the second axis is again of order 5, there are 4 stripes both above and below the diagonal with the fifth strip coinciding with the diagonal.

Special care has to be taken when applying the zero boundary condition. In the one-dimensional case, the border of the matrix is simply removed and the dimension of the problem is reduced from N^2 to $(N - 2)^2$. For the multidimensional case, the border points must be removed for each axis separately and the corresponding rows and columns must be removed from inside the total matrix.

Luckily this is very simple to practically implement when constructing the matrix from scratch. An example with the same parameters as before is shown in the density graph 2.5. The length of the side of the matrix is reduced to $(13 - 2)(17 - 2) = 165$.

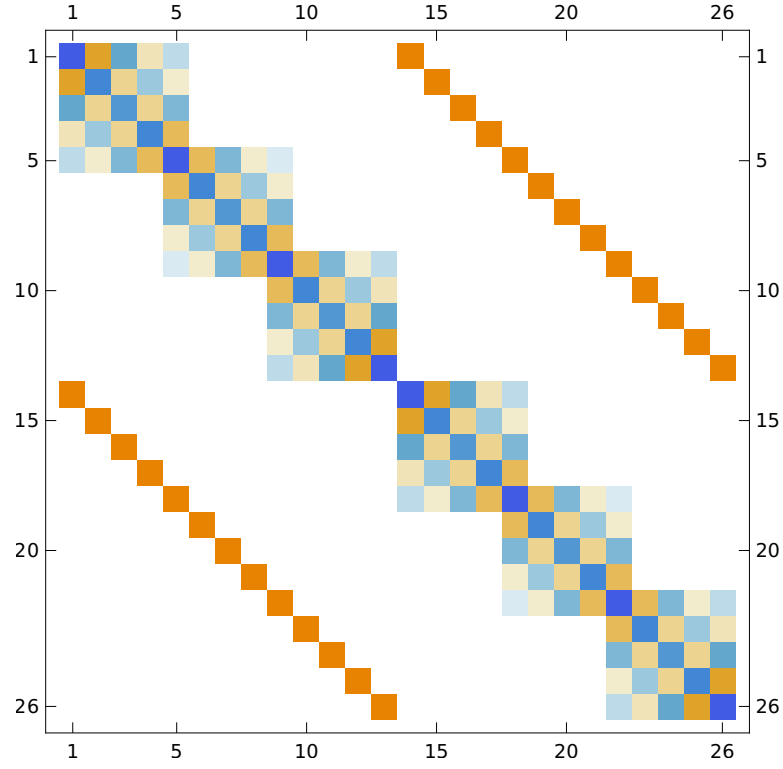


Figure 2.3 The sparsity pattern of a two-channel Hamiltonian H for $G = 5$ and $N = 3$.

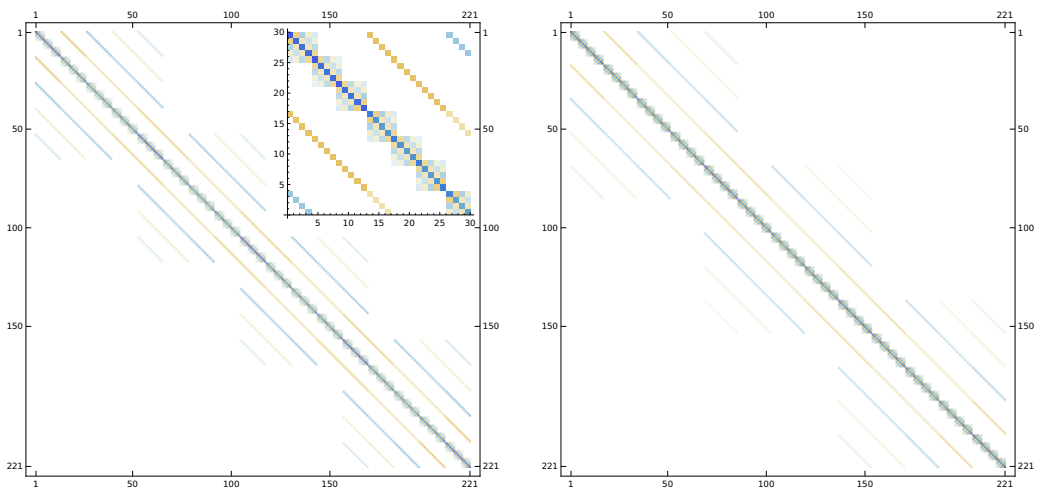


Figure 2.4 The sparsity pattern of a two-dimensional Hamiltonian H for $G_1 = G_2 = 5$, $N_1 = 3$ and $N_2 = 4$ for ordering 12 (left) and 21 (right).

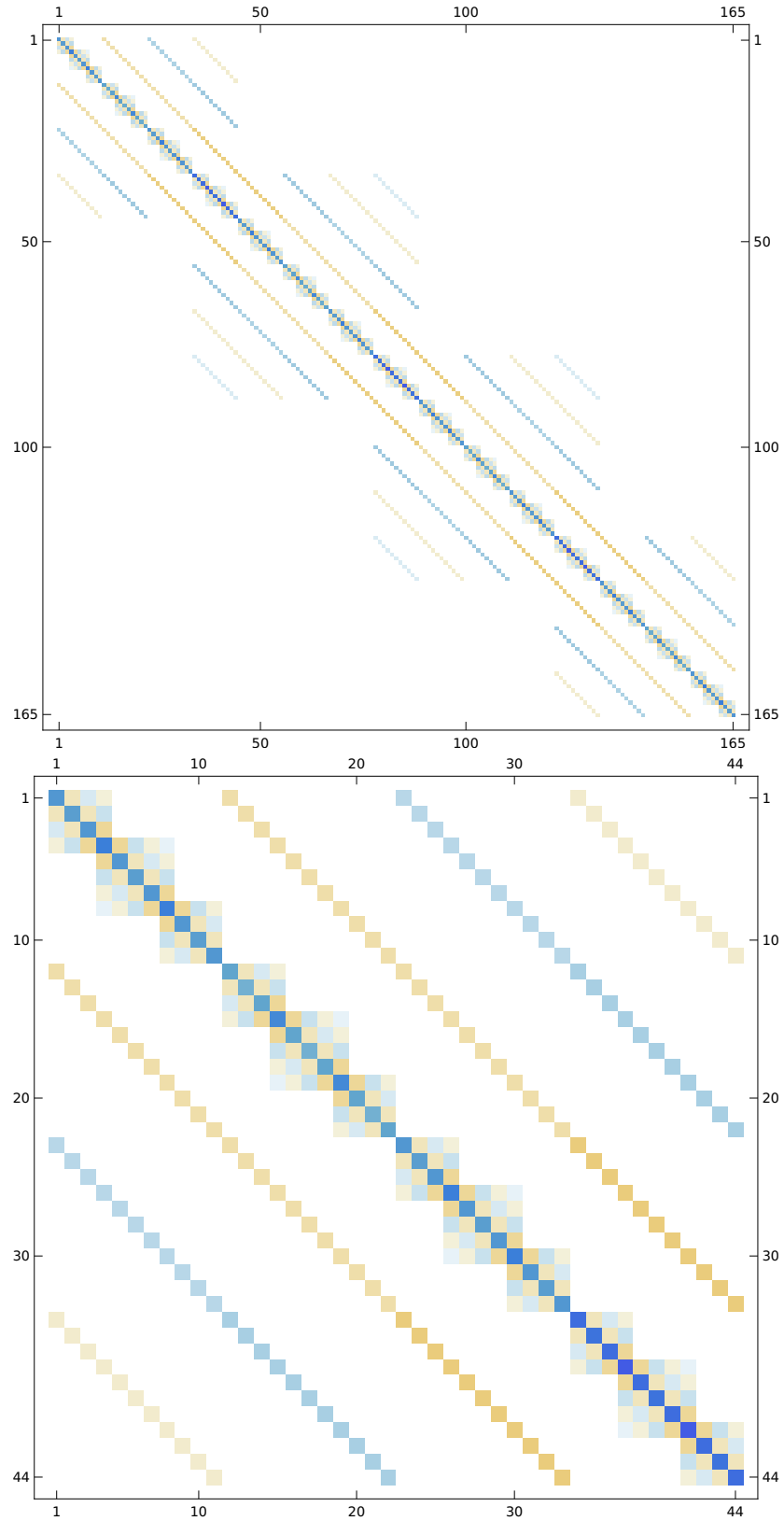


Figure 2.5 The sparsity pattern of a borderless two-dimensional Hamiltonian H for $G_1 = G_2 = 5$, $N_1 = 3$ and $N_2 = 4$ for ordering 12 (top) and a zoomed-in portion (bottom) showing the skipped rows and columns in the inner one-dimensional stiffness matrix structure.

Chapter 3

Results and benchmarks

A COCR solver using the FEM-DVR-ECS method described in chapter 2 was written in the Rust programming language using the nightly toolchain. For basic linear algebra, the *nalgebra* 0.30.1, *nalgebra-sparse* 0.6.0 and *nalgebra-lapack* 0.21.0 crates were utilized. Intel MKL 2020.4.304 was chosen as the *Lapack* implementation. Parallel computations were performed via the *rayon* 1.5.3 crate. The entire codebase is available in [12]. A few implementation details are mentioned in the appendix B.

For the Gauss-Lobatto quadrature points and weights, an existing FORTRAN implementation was linked, as well as for the Lagrange polynomial derivatives. The method labeled as *direct* refers to the *zsysv* method from the Intel MKL Lapack library for the two-channel one-dimensional case and to the *Intel Direct Sparse Solver* (also linked from the specified Intel MKL library) in the two-dimensional model case.

Next, both models from chapter 1 were implemented. All simulations were run on a device with relevant specifications in the table 3.1

Kernel	Linux 5.18.3-arch1-1
CPU	AMD Ryzen 9 5950X
SMT	Enabled - 32 threads
RAM	64 GiB @ 3466MHz DDR4

Table 3.1 Relevant software and hardware specifications of the system used for all computations

All runtime graphs are of tentative character. Runtime is not only affected by raw hardware performance (CPU cache, memory clock speed, etc.), but also by available and enabled CPU features, such as AVX256. All benchmark programs were compiled with all available features enabled using the rustc *-C target-cpu=native* compiler flag. There are also factors of stochastic nature, such as issues with CPU scheduling, other processes stealing system resources and similar effects. These could be eliminated by performing statistics on the runtime for each set of input parameters. This is however costly and tentative results are sufficient for discussion.

3.1 The two-channel model

This section displays the results for the model introduced in 1.1. The results of this model show general features of the chosen iterative method. They also provide a way to demonstrate quantities of interest and a few ways to plot them.

Each run is parametrized by the initial energy of the electron E_{el} , the initial state $\psi_{0,i}$, which determines the right-hand side vector b and the Gauss-Lobatto quadrature order G , which determines the dimension of the finite approximation. The elements remained the same for all simulations, there are 2 real elements in the interval $[0, 2]$ and 4 ECS elements up to $R_{\text{max}} = 2 + 28e^{i\frac{\pi}{4}}$. All points are shown in detail in the appendix A.

The output of each run is the number of iterations $\mathcal{N}_t$ needed to reach RHS-relative residuum $\xi < \xi_t = 10^{-8}$, the runtime t (including the time needed to calculate the preconditioner), the RHS-relative residuum $\xi < \xi_t$ and the calculated cross-section $\sigma_{i \rightarrow j}$. Relating the input and output parameters, we get the relation

$$(G, E_{\text{el}}, i) \mapsto (\mathcal{N}_t, t, \xi, \sigma_{i \rightarrow j}) \quad (3.1)$$

for quantities related to the entire run. All the output quantities except the runtime t are independent on the system and configuration.

If we pick a specific run (G, E_{el}, i) , we can measure an additional dependence of the RHS-relative residuum $\xi(\mathcal{N})$ on the number of iterations $\mathcal{N}$ inside the run. This provides a way of observing the convergence properties of selected methods. We refer to this kind of plots as convergence graphs.

The differences of quantities between initial states $\psi_{0,1}$ and $\psi_{0,2}$ are rather uninteresting and only the results for $\psi_{0,1}$ are plotted. In the physically interesting region ($E_{\text{el}} < 3$), the absolute relative difference $|\Delta\mathcal{N}_t|/\max(\mathcal{N}_{t,1}, \mathcal{N}_{t,2})$, $\Delta\mathcal{N}_t = \mathcal{N}_{t,1} - \mathcal{N}_{t,2}$, is mostly below 10% for $G = 15$, as is seen in the graph 3.1.

3.1.1 Results

The interval $E_{\text{el}} \in (0, 5)$ was sampled using 1024 points for Gauss-Lobatto orders $G \in \{5, 10, 15\}$ and $i = 1$. The dependence of runtime $t(E_{\text{el}})$ on the initial electron energy for all methods (including a direct method) is shown in the graph 3.2. Since the unpreconditioned iterative method is worse than ILU0 and the direct method by an order of magnitude, the two best methods are separately plotted in the graph 3.3. The iterative methods are also compared by the target number of iterations $\mathcal{N}_t(E_{\text{el}})$ in the graph 3.4.

Convergence graph 3.5 was plotted for energies $E_{\text{el}} \in \{0.1, 1.9, 2.1, 3.0\}$. This choice was made to cover low energies, high energies and energies just before and after the second channel opens at $E_{\text{el}} = 2$. ILU0 dominates the other iterative methods in both the runtime and the number of iterations, losing only to the direct method, but still manages to operate in the same order of runtime.

The difference between the ILU0 preconditioned iterative solution for cross-section and the analytic results $\varepsilon_{a,i \rightarrow f} = |\sigma_{i \rightarrow f} - \sigma_{a,i \rightarrow f}|$ is shown in the graph 3.6. The order of the difference from the analytic solution is the largest around the thresholds and the resonance. This can be improved by prolonging the ECS region to more accurately calculate slowly oscillating and decreasing wave functions (low electron momentum). This lowers the error around thresholds and makes the error

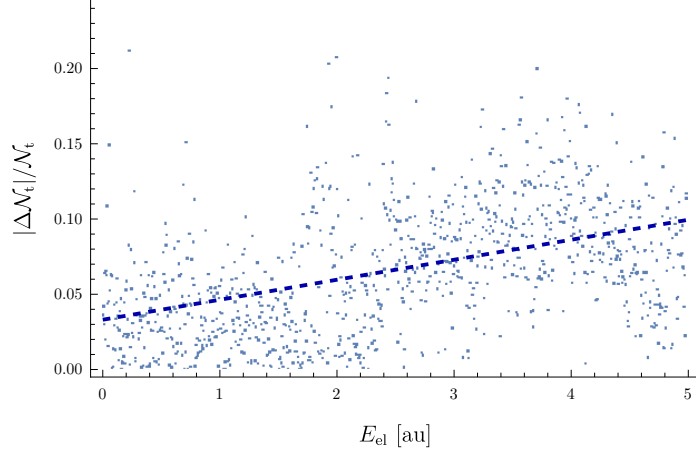


Figure 3.1 The absolute relative difference of numbers of required iterations $\mathcal{N}_t$ needed to reach the target RHS-relative residuum $\xi_t = 10^{-8}$ for initial states $\psi_{0,1}$ and $\psi_{0,2}$ and $G = 15$ as a function of the initial electron energy E_{el} . The dashed line is a linear fit of the values.

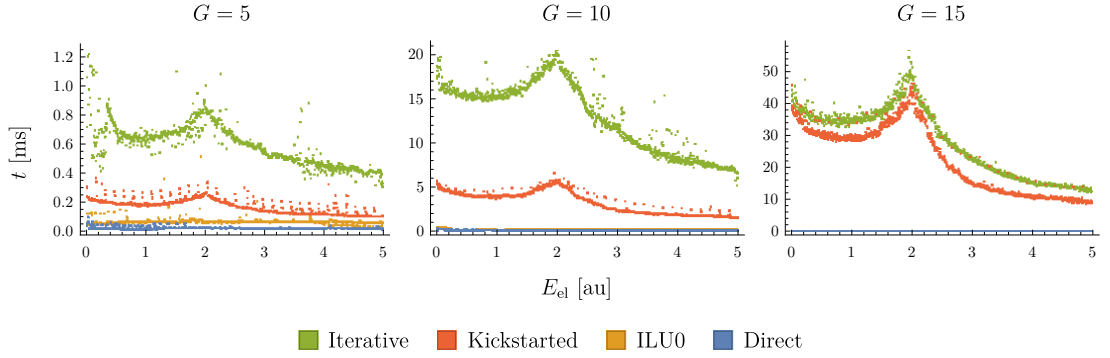


Figure 3.2 Runtime t of the two-channel model for all methods as a function of the initial electron energy E_{el} for target $\xi_t = 10^{-8}$ and the initial state $i = 1$. The unpreconditioned iterative method is significantly influenced by the presence of thresholds.

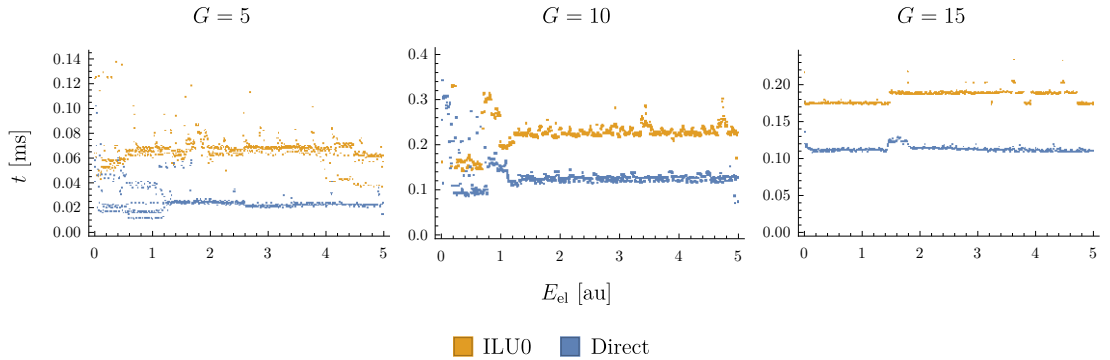


Figure 3.3 Runtime t of the two-channel model for ILU0 and the direct method as a function of the initial electron energy E_{el} for target $\xi_t = 10^{-8}$ and the initial state $i = 1$. The thresholds and the resonance do not affect the runtime dramatically.

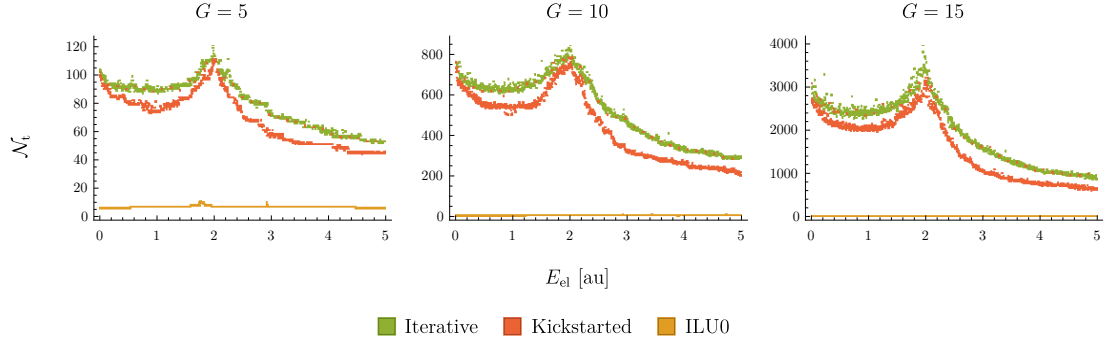


Figure 3.4 Number of iterations $\mathcal{N}_t$ of the two-channel model needed to reach the RHS-relative residuum $\xi_t = 10^{-8}$ for all iterative methods as a function of the initial electron energy E_{el} and the initial state $i = 1$.

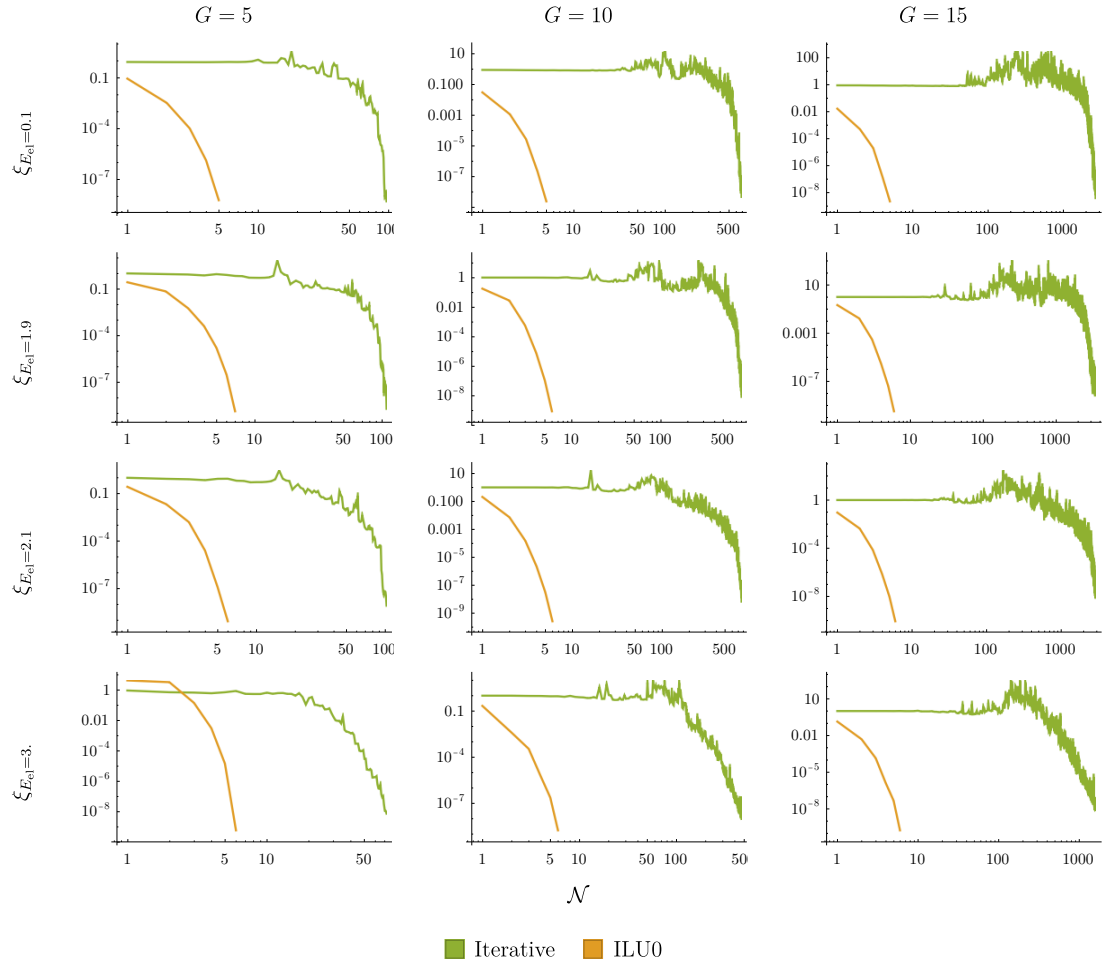
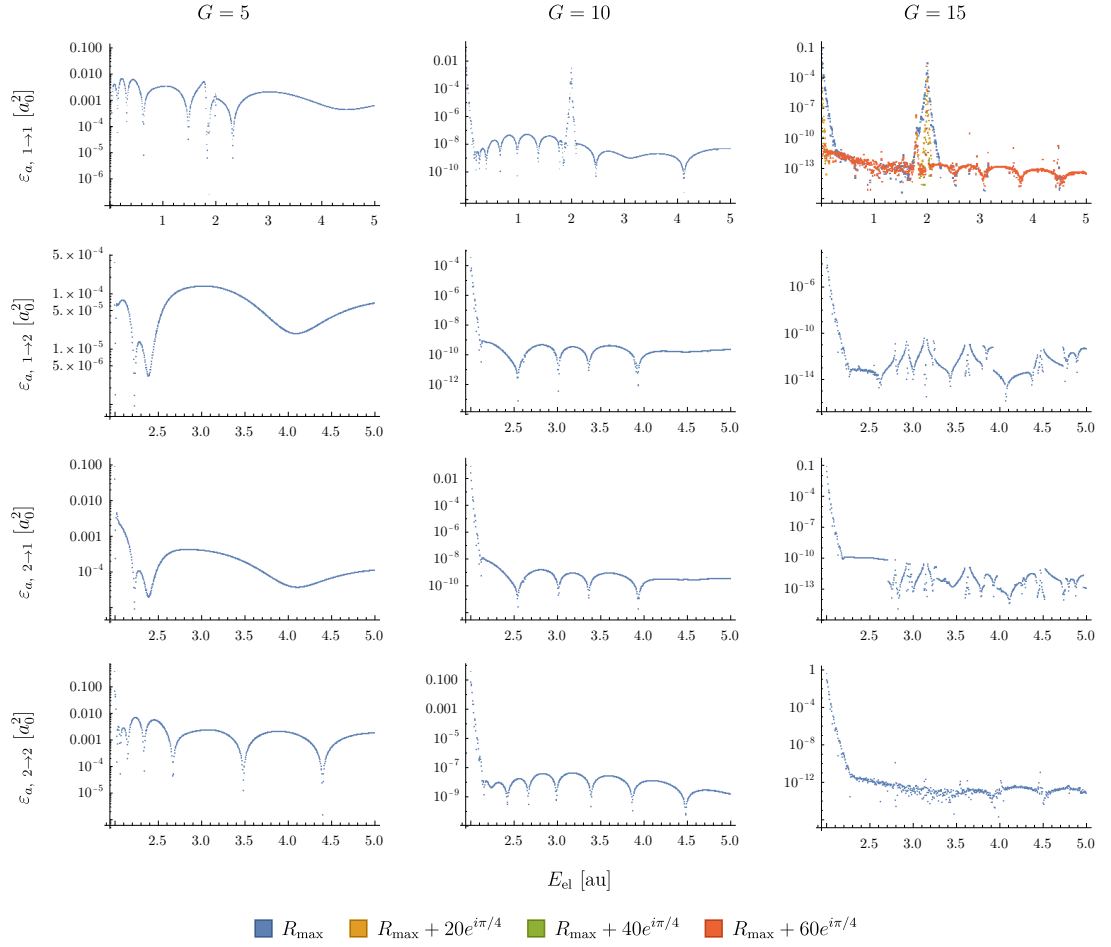


Figure 3.5 The RHS-relative residuum ξ of the two-channel model as a function of iteration steps $\mathcal{N}$ for initial electron energies $E_{\text{el}} \in \{0.1, 1.9, 2.1, 3.0\}$ and the initial state $i = 1$.

near the resonance more apparent. The difference gets smaller with higher orders of the Gauss-Lobatto quadrature as is expected.

The effect of E_{el} sampling density on kickstarting is shown in the graph 3.7. It does not seem to provide a significant improvement in the number of target iterations and for this reason it was not studied any further or at all in the context of the two-dimensional model.



$1 \rightarrow 1$

$G = 15$

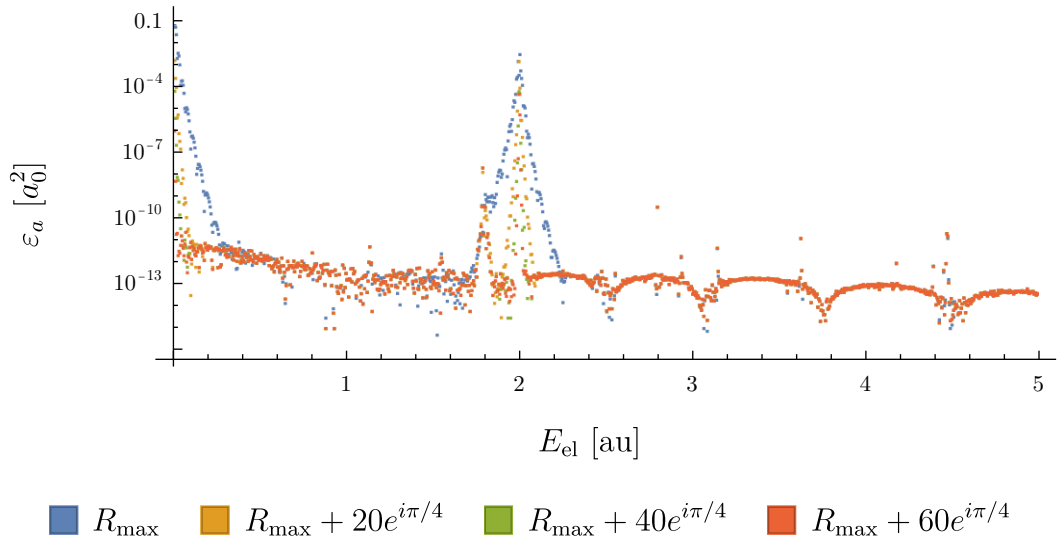


Figure 3.6 The absolute difference ε_a of the ILU0 preconditioned iterative and analytic solutions for the cross-section $\sigma_{i \rightarrow j}$ for target RHS-relative residuum $\xi = 10^{-8}$ as a function of the initial electron energy E_{el} . The inaccuracy around thresholds can be eliminated by prolonging the ECS part of the grid, as is shown for the $1 \rightarrow 1$ transition and $G = 15$.

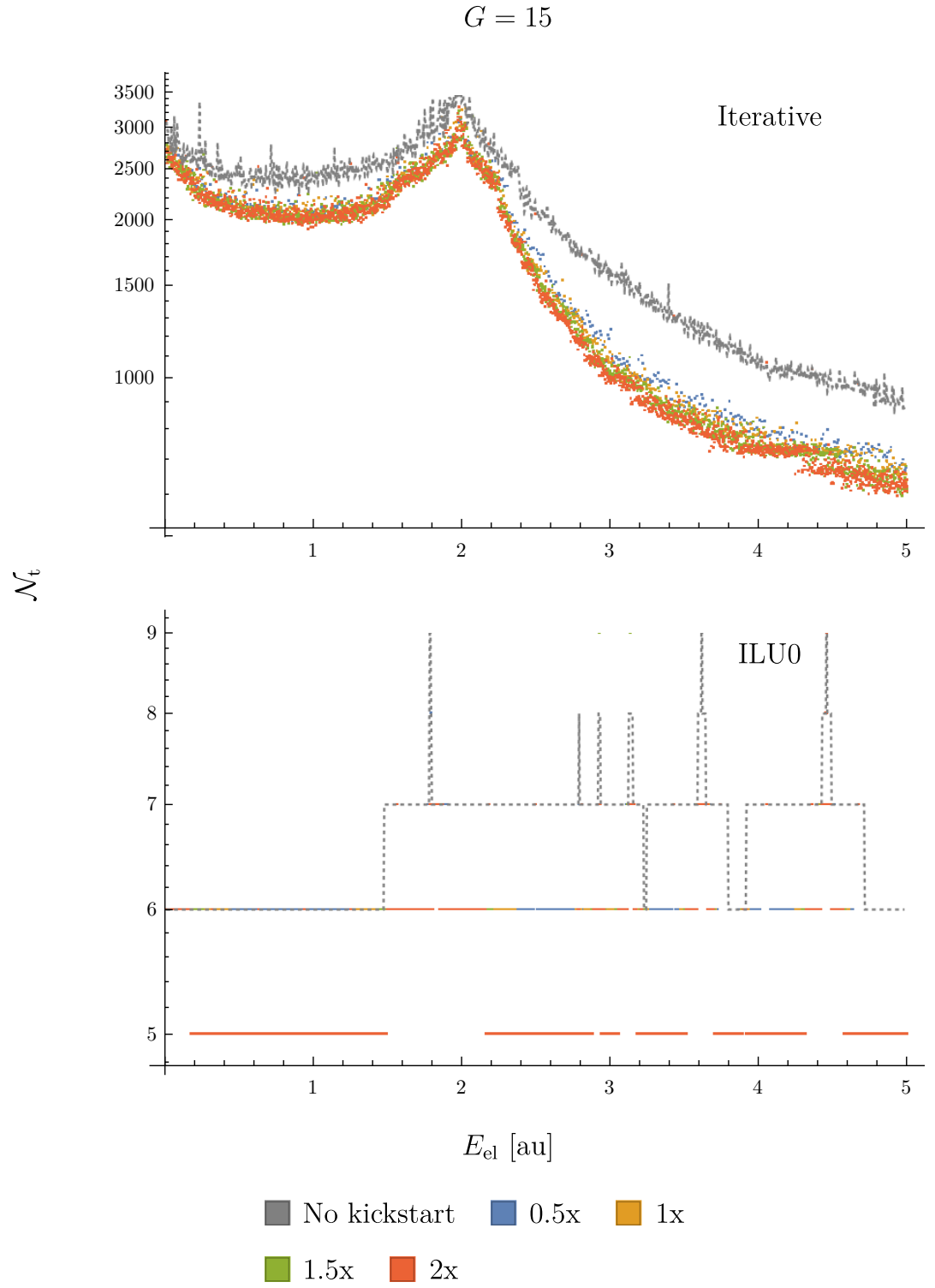


Figure 3.7 The effect of E_{el} point sampling density on the efficiency of kickstarting. The effect of increased density seems to be negligible for both the unpreconditioned and the ILU0 preconditioned iterative method. Kickstarting however is almost always faster than not kickstarting and the cumulative gain for simulations run on a single thread can be significant.

3.2 The 2D model

For the 2D model introduced in 1.2, the input parameter space now has a Gauss-Lobatto quadrature order for each coordinate independently and also the ordering parameter discussed in 2.3.2. To make the notation more understandable, we use the subscripts e and m for each coordinate instead of notation using digits 1 and 2. The electronic grid is of Gauss-Lobatto order G_e , and corresponds to the coordinate r from theory. The molecular grid is of order G_m and corresponds to the R coordinate. The two-dimensional grid is constructed using the two orderings described in 2.3.2 denoted me and em . To plot energy dependence we, again, use the initial electron energy E_{el} .

The input and output parameter mapping is

$$(G_m, G_e, me/em, E_{\text{el}}, i) \mapsto (\mathcal{N}_t, t, \xi, \sigma_{VE}, i \rightarrow j, \sigma_{DA}) \quad (3.2)$$

for each model and numerical method available, where $\xi < \xi_t = 10^{-8}$. The number of sampled points E_{el} depends on the order of the quadrature and is equal to 512, 256, 128, and 64 for (G_m, G_e) equal to (5, 5), (10, 8), (15, 11), and (17, 15). The only exception is for F_2 where the number of sampled points is always equal to 32.

For each input parameter tuple we still have the RHS-relative residuum size $\xi(\mathcal{N})$ dependence available and most of the analysis shown in the one-dimensional case can also be done here, except for the comparison with the analytic solution. This type of analysis is replaced with a comparison against a result obtained with the direct method and a sufficiently large Gauss-Lobatto quadrature order $(G_m, G_e) = (17, 15)$. This absolute difference is denoted $\varepsilon_{h, i \rightarrow j}$ (where the h stands for highest order).

The parameters of all grids are shown in the table 3.2. A table of exact elements for each model is available in the appendix A.

Diatomic	Grid	N_{real}	N_{ECS}	R_0	R_{ECS}	R_{max}
N_2	m	10	6	0	6	$6 + 100e^{i\frac{\pi}{4}}$
	e	12	6	0	50	$50 + 100e^{i\frac{2\pi}{9}}$
NO	m	15	6	0	8.5	$8.5 + 100e^{i\frac{\pi}{4}}$
	e	12	6	0	50	$50 + 100e^{i\frac{2\pi}{9}}$
F_2	m	80	40	2	10	$10 + 15e^{i\frac{\pi}{4}}$
	e	13	6	0	64.5	$64.5 + 100e^{i\frac{2\pi}{9}}$
O_2	m	47	0	0	-	15
	e	22	8	0	71	$71 + 200e^{i\frac{2\pi}{9}}$

Table 3.2 Parameters of grids of the two-dimensional model for each diatomic. N_{real} is the number of real elements, N_{ECS} is the number of ECS elements, R_0 is the initial point of the grid, R_{ECS} is the point where ECS starts and R_{max} is the final point of the grid.

3.2.1 Convergence properties

As was the case with the one-dimensional model, the unpreconditioned iterative method has a very poor convergence rate and for this reason it was benchmarked only for the case of the two lowest quadrature orders with N_2 . The runtime and

number of iterations required to reach RHS-relative residuum $\xi_t = 10^{-8}$ for N_2 , NO, O_2 and F_2 are shown in the graphs 3.10, 3.12, 3.14 and 3.16. Convergence graphs at preselect energies are shown for N_2 , NO and O_2 in the graphs 3.11, 3.13 and 3.15. The F_2 calculations failed to converge in a preset maximal amount of steps (referred to as the *iteration limit*) for the higher quadratures and even at $E_{el} > 0.05$ for the lowest quadrature, therefore the results are of tentative character. The iteration limit is decreasing with quadrature order and is equal to 100, 65, 25 and 10 thousand iterations, to ensure that even the most demanding simulation points finish in at most a few hours. Typically, the runtimes ranged from a few milliseconds for lower quadrature orders with ILU0 to hours for partial inverse and higher quadratures. The minimum runtime for an iterative solver to reach $\xi_t = 10^{-8}$ was 14.43 ms (N_2 , ILU0 (em), $(G_m, G_e) = (5, 5)$, $\mathcal{N}_t = 48$) and the maximum, that was obtained before the mentioned iteration limit was set, was 5.18 h (NO, partial inverse (me), $(G_m, G_e) = (15, 11)$, $\mathcal{N}_t = 51118$).

ILU0 dominates when it comes to both the number of iterations and the total runtime. Only for the case of N_2 the em ordering variant of partial inverse competes with it in terms of iteration count, where it manages to outperform it at higher energies. This is best seen in the convergence graphs 3.11. When it comes to runtime, the method is however much worse. This has to do with both the cost of computing the preconditioner (direct inverses of blocks similar to 1D stiffness matrices) and the number of multiplications per iteration (equivalently, the number of non-zero elements of the preconditioned matrix). The ordering of the grid is only of significance for the partial inverse method. For ILU0 and the unpreconditioned iterative method, the effect of ordering is negligible, and the convergence graphs are nearly indistinguishable.

All the methods display decreasing convergence rate with increasing kinetic energy. This makes the iterative approach difficult to use to compute the dissociative attachment cross-section σ^{DA} , because the threshold is usually located at higher energies. Although the energy dependence $\mathcal{N}_t(E_{el})$ shows a particular trend, many points distributed randomly across the studied ranges show unusually high iteration counts, inability to converge below some value of ξ or even divergence after a few thousand iterations. The points of divergence are shown in the iteration graphs as dashed gray vertical lines.

From the collected data it is apparent, that ILU0 is the preconditioner most prone to this divergence phenomenon. It is most pronounced in the case of O_2 with quadrature $(G_m, G_e) = (10, 8)$. The divergence points are shown in the table 3.3. Most points diverge for both orderings, but there are two points, where the em ordering converges and the seemingly less efficient ordering me does not. It is also evident, that the me ordering diverges a couple of hundred iterations faster. ILU0 also happens to diverge for N_2 for the two lowest quadrature orders, for the order $(G_m, G_e) = (10, 8)$ it happened at $E_{el} \approx 120.547 \times 10^{-3}$ au again only for the me ordering after 54464 iterations.

A neighborhood of one point of divergence and a convergence plot is shown in the graph 3.8. There appears to be a narrow plateau in the graph of the number of iterations in this region. At a given point of this plateau, the RHS-relative residuum ξ gets stuck after a certain number of iterations close to what would have been sufficient for the calculation to hit the target ξ_t , and then diverges after a few thousand additional steps. The plateau for the em ordering spans

the range 4.9707×10^{-3} AU to 4.97537×10^{-3} AU and is narrower than for the me ordering, where it spans the range 4.97004×10^{-3} AU to 4.97587×10^{-3} AU (25 % wider).

approx. $E_{\text{el}} [10^{-3} \text{ au}]$	$\mathcal{N}_{\text{div},me}$	$\mathcal{N}_{\text{div},em}$
2.949	2724	2834
4.971	4659	-
9.014	4700	4977
11.260	5464	5787
12.158	6546	6705
19.346	9398	-

Table 3.3 The points of divergence of the ILU0 method for O_2 and Gauss-Lobatto quadrature $(G_m, G_e) = (10, 8)$. The hyphen symbols signifies that the calculation converged.

When it comes to the other preconditioners, the only case of divergence detected in the examined range happened for partial inverse in the em ordering for NO with order (15, 11) at $E_{\text{el}} \approx 17.109 \times 10^{-3}$ au after 11909 iterations.

The Intel Direct Sparse Solver defeated ILU0 in runtime for all calculations. The ratio of their runtimes is shown in the graph 3.9. The direct solver however requires much more memory. This was most noticeable for O_2 with $(G_m, G_e) = (17, 15)$, where 32 threads would quickly fill up all available RAM and kernel would then kill the process. To perform these calculations, the number of threads had to be lowered to 8. A brief review of the GNU time[13] command output for calculation of one point E_{el} shows the ILU0 method consuming 1.05 GiB of memory and the DSS method consuming 2.4 GiB (129 % more).

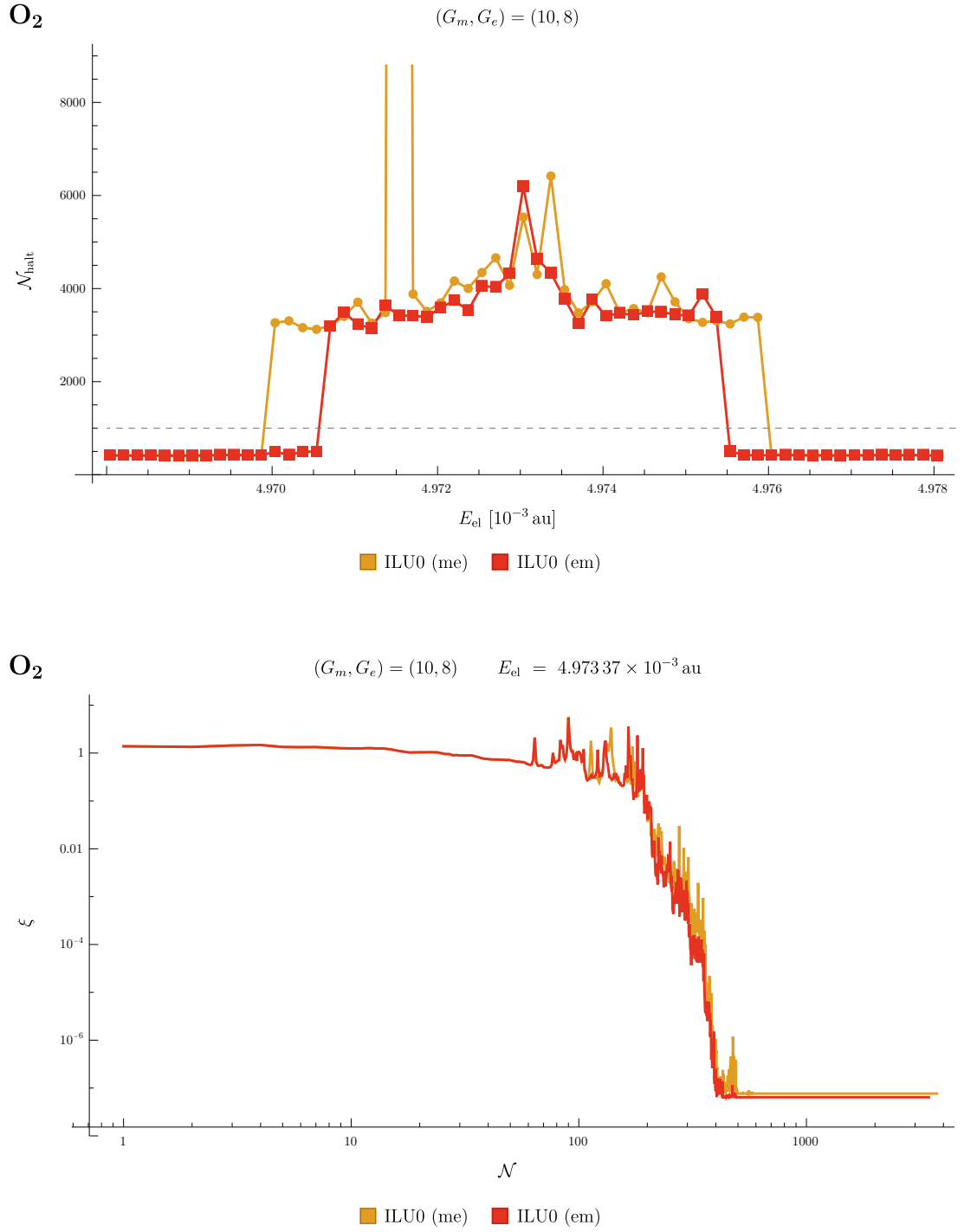


Figure 3.8 The number of iterations $\mathcal{N}_{\text{halt}}$ required for the calculation to either converge (points under the dashed line) or diverge (points above the dashed line) in the region of divergence of the ILU0 method for O_2 and Gauss-Lobatto quadrature $(G_m, G_e) = (10, 8)$ and a convergence graph in one of the points inside this region. The plateau of divergence is narrow for the em ordering and slightly wider for the me ordering. At one point of the me plateau, the calculation hit the iteration limit of 90000.

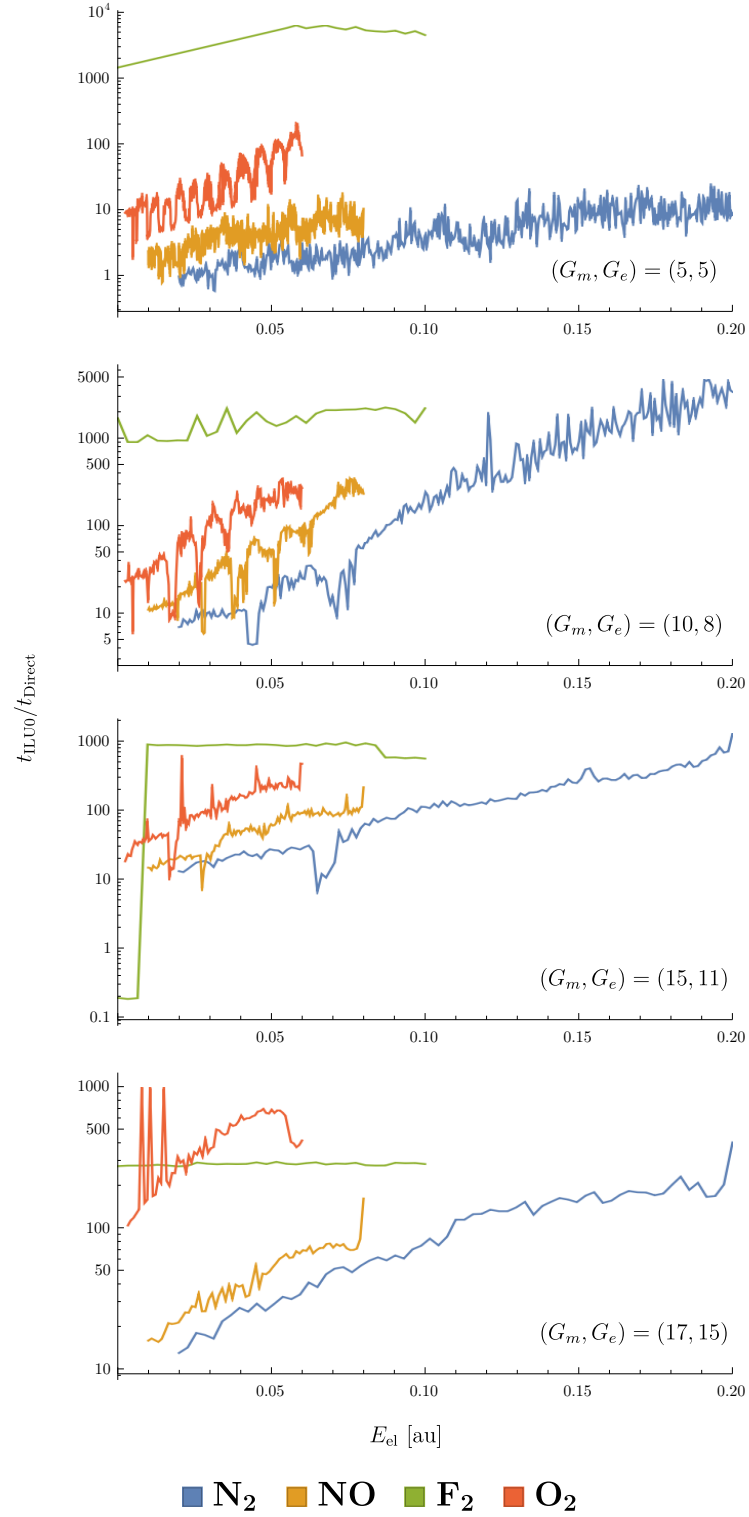


Figure 3.9 The ratio of the runtimes t of the ILU0 method with target $\xi_t = 10^{-8}$ and the Intel Direct Sparse Solver for four different Gauss-Lobatto quadrature orders as a function of the initial electron energy E_{el} . The flatness of the curve for F_2 is caused by the iteration limit (decreasing with quadrature: 100k, 65k, 25k, 10k). The limit is also hit by O_2 at certain lower energies for the higher quadrature orders.

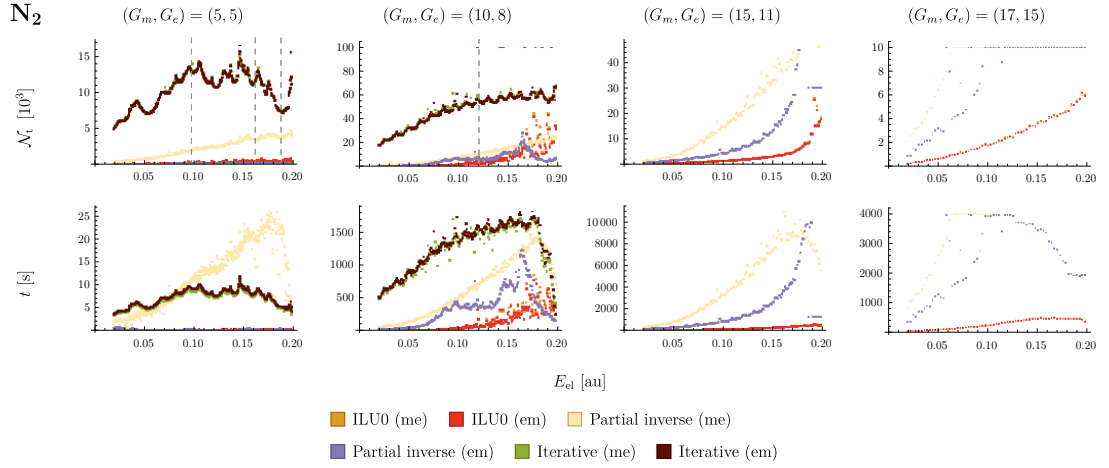


Figure 3.10 The dependence of target iterations count $\mathcal{N}_t$ (top row) and runtime t (bottom row) on initial electron energy E_{el} for the diatomic N_2 , 4 different Gauss-Lobatto quadrature orders (horizontal axis) and target $\xi_t = 10^{-8}$. The gray dashed vertical lines signify points, where the simulation diverged for ILU0. The unpreconditioned iterative method is only shown for the first two quadrature orders, because of the iteration limit. This limit is also hit by the partial inverse method for both orderings in the higher quadrature orders.

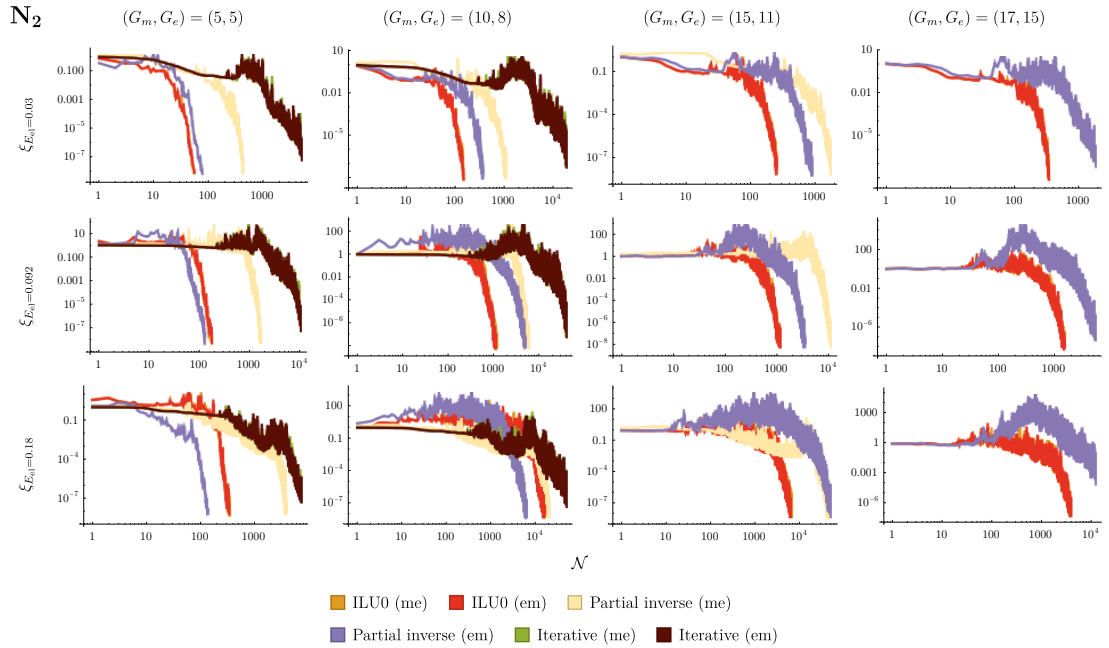


Figure 3.11 The convergence graph $\xi(\mathcal{N})$ for the diatomic N_2 , 4 different Gauss-Lobatto quadrature orders (horizontal axis), 3 preselect energies $E_{\text{el}} \in \{0.03, 0.092, 0.18\}$ (vertical axis) and different methods and grid orderings plotted up to target $\xi_t = 10^{-8}$ or the iteration limit. The unpreconditioned iterative method is only shown for the first two quadrature orders, because the number of required steps to hit the target grows significantly with the quadrature order.

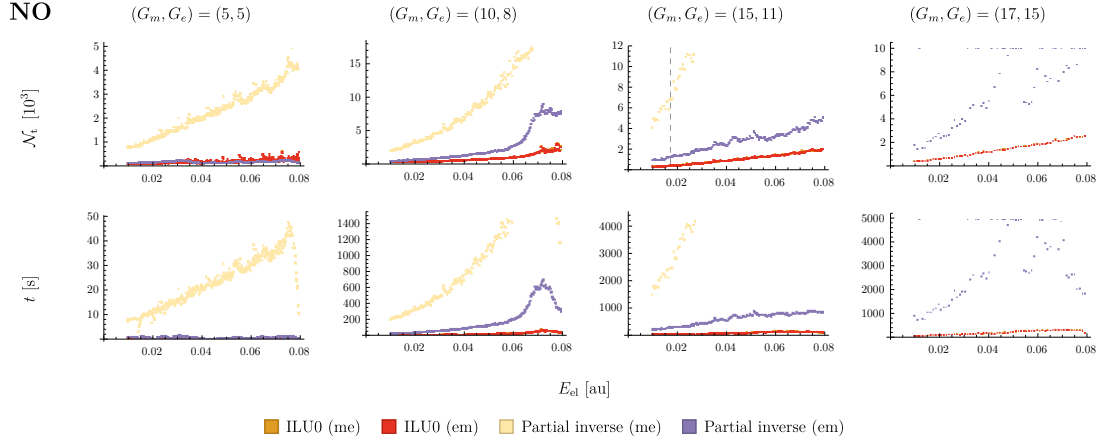


Figure 3.12 The dependence of target iterations count $\mathcal{N}_t$ (top row) and runtime t (bottom row) on initial electron energy E_{el} for the diatomic NO, 4 different Gauss-Lobatto quadrature orders (horizontal axis) and target $\xi_t = 10^{-8}$. The gray dashed vertical line signifies the point, where the simulation diverged for the em ordered partial inverse method. The partial inverse method in the me ordering is omitted for the highest quadrature order, as it already hit the iteration limit in the lower order.

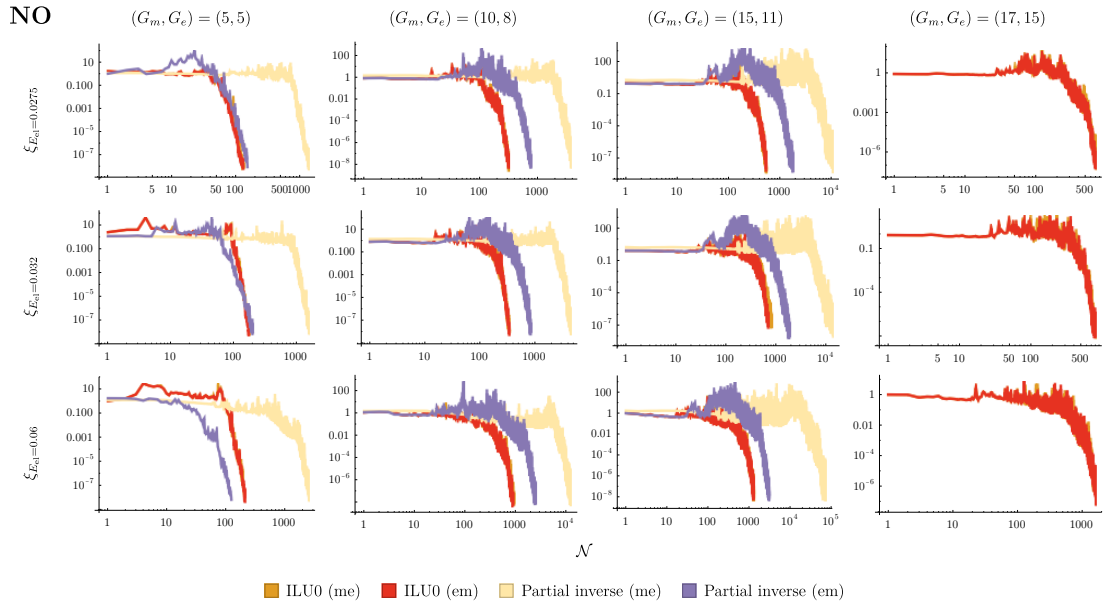


Figure 3.13 The convergence graph $\xi(\mathcal{N})$ for the diatomic NO, 4 different Gauss-Lobatto quadrature orders (horizontal axis), 3 preselect energies $E_{el} \in \{0.0275, 0.032, 0.06\}$ (vertical axis) and different methods and grid orderings plotted up to target $\xi_t = 10^{-8}$ or the iteration limit. Partial inverse was not tested for the highest quadrature order, because it is inefficient.

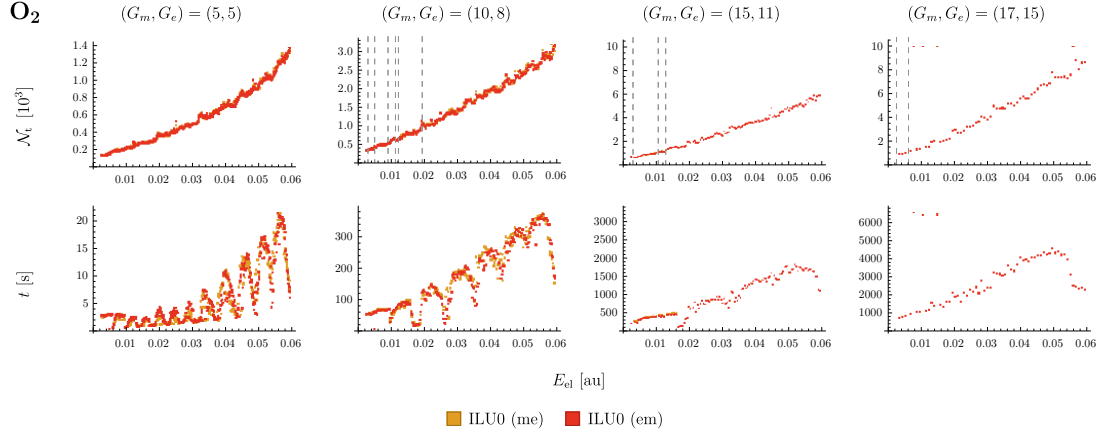


Figure 3.14 The dependence of target iterations count $\mathcal{N}_t$ (top row) and runtime t (bottom row) on initial electron energy E_{el} for the diatomic O_2 , 4 different Gauss-Lobatto quadrature orders (horizontal axis) and target $\xi_t = 10^{-8}$. The gray dashed vertical lines signify points, where the simulation diverged for ILU0.

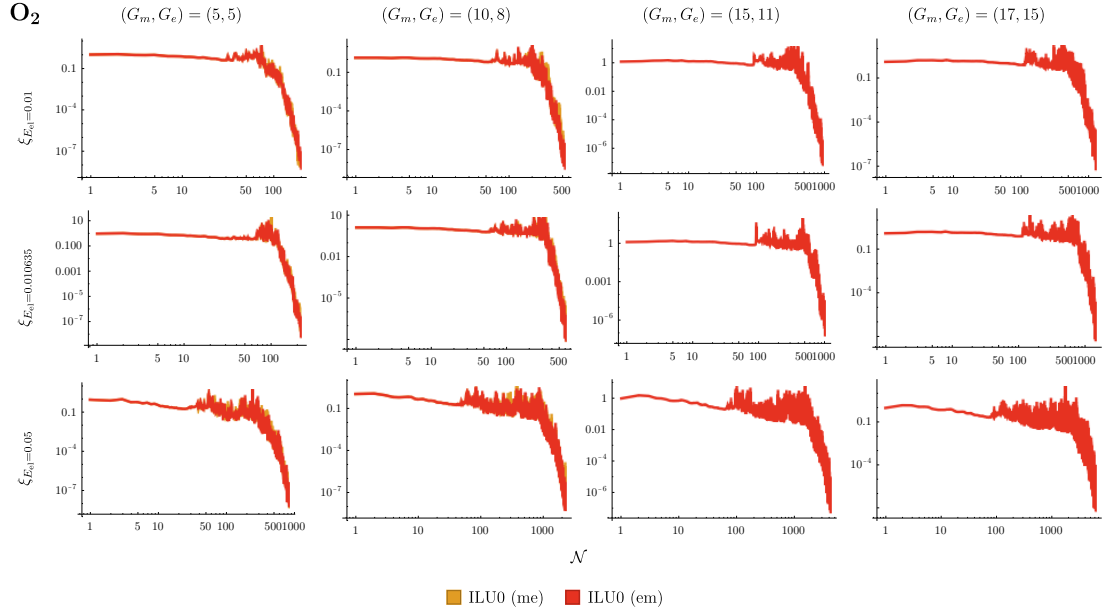


Figure 3.15 The convergence graph $\xi(\mathcal{N})$ for the diatomic O_2 , 4 different Gauss-Lobatto quadrature orders (horizontal axis), 3 preselect energies $E_{el} \in \{0.01, 0.010635, 0.05\}$ (vertical axis) and different methods and grid orderings plotted up to target $\xi_t = 10^{-8}$ or the iteration limit.

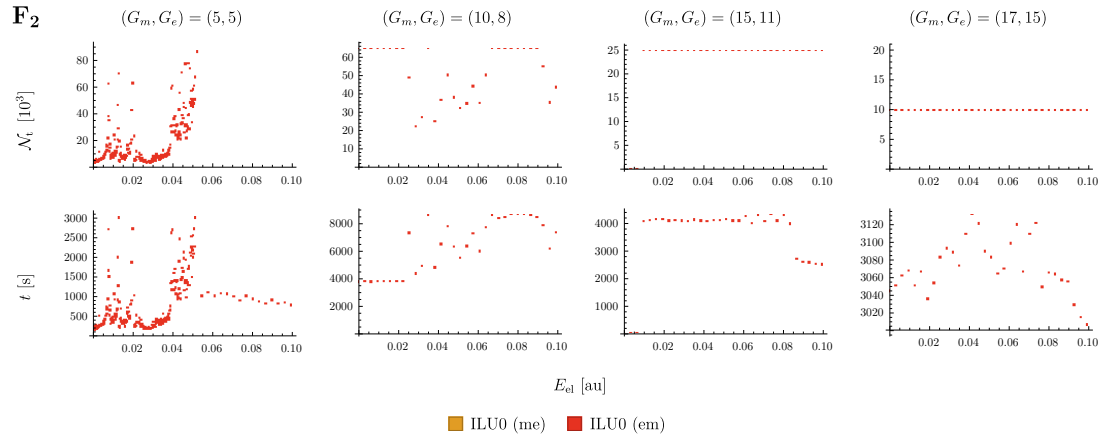


Figure 3.16 The dependence of target iterations count $\mathcal{N}_t$ (top row) and runtime t (bottom row) on initial electron energy E_{el} for the diatomic F_2 , 4 different Gauss-Lobatto quadrature orders (horizontal axis) and target $\xi_t = 10^{-8}$. Only the lowest Gauss-Lobatto quadrature order gives results in a reasonable time. The higher orders consistently hit their iteration count limits.

3.2.2 The cross-section σ^{VE}

The resulting vibrational excitation cross-sections $\sigma^{\text{VE}}(E_{\text{el}})$ for the diatomics N_2 , F_2 , NO and O_2 are shown in the graphs 3.17, 3.18, 3.19 and 3.20. Absolute differences ε_h of solutions of a lower Gauss-Lobatto quadrature order and the highest used order $(G_m, G_e) = (17, 15)$ are shown in the graph 3.21. The obtained results for the first three molecules are in agreement with the data provided in[4].

The results for O_2 are always ahead in terms of accuracy when compared to the other systems. This is because the number of elements (N_m, N_e) for this molecule is higher, while also converging under the iteration limit unlike F_2 . The dependence on the number of elements can be studied further by choosing a reference initial electron energy E_{el} and plotting the dependence of the absolute difference from the highest order solution ε_h on the number of molecular grid points N_m . For N_2 such dependence is shown in the graph 3.22, where the grid is chosen to be equidistant on the range $R \in [0; 6]$ and the reference initial energy is $E_{\text{el}} \approx 0.079 \text{ au}$.

On the other hand, the results for F_2 are not converging to the direct solution fast enough and the iteration limit is almost consistently hit. This, however, does not affect the vibrational excitation cross-sections as much as the dissociative attachment cross-section.

3.2.3 The cross-section σ^{DA}

The dissociative attachment cross-section σ^{DA} was calculated only for the case of $\nu_i = 0$ and F_2 , since for the other molecules this channel opens at much higher energies than the sampled intervals cover. The results of the ILU0 preconditioned iterative method and the direct method are shown in the graph 3.23. Since most of the points did not converge under the iteration limit, the iterative results are very inaccurate. Unlike the VE channel, the dissociative attachment cross-section is affected much more significantly to the point where the calculated results provide no usable information.

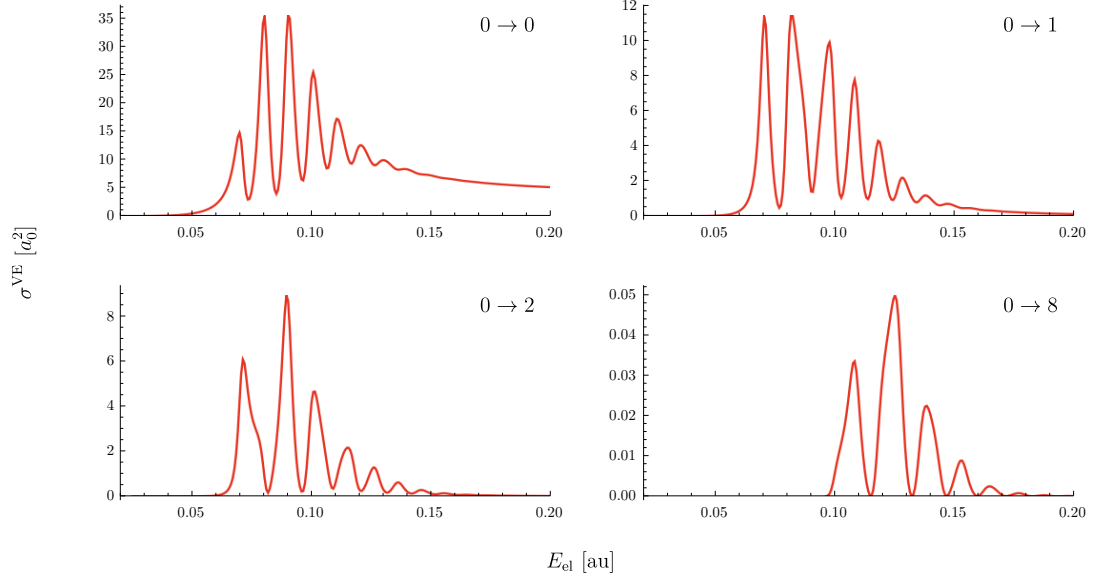
N₂ $(G_m, G_e) = (10, 8)$ 

Figure 3.17 The cross-section $\sigma^{\text{VE}}(E_{\text{el}})$ dependence on initial electron energy E_{el} obtained for the diatomic N_2 via the ILU0 preconditioned iterative method ($\xi_t = 10^{-8}$), 4 different transitions and Gauss-Lobatto quadrature of order $(G_m, G_e) = (10, 8)$.

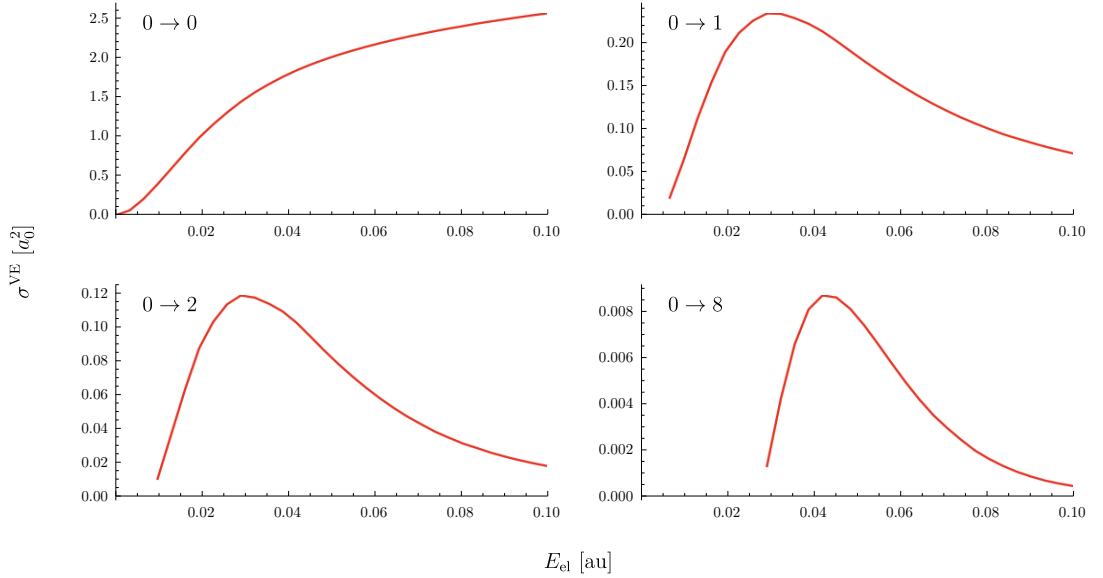
F₂ $(G_m, G_e) = (10, 8)$ 

Figure 3.18 The cross-section $\sigma^{\text{VE}}(E_{\text{el}})$ dependence on initial electron energy E_{el} obtained for the diatomic F_2 via the ILU0 preconditioned iterative method ($\xi_t = 10^{-8}$), 4 different transitions and Gauss-Lobatto quadrature of order $(G_m, G_e) = (10, 8)$. Most of the points $E_{\text{el}} > 0.5$ did not converge to ξ_t below the preset number of iterations, but are accurate enough.

NO

$$(G_m, G_e) = (10, 8)$$

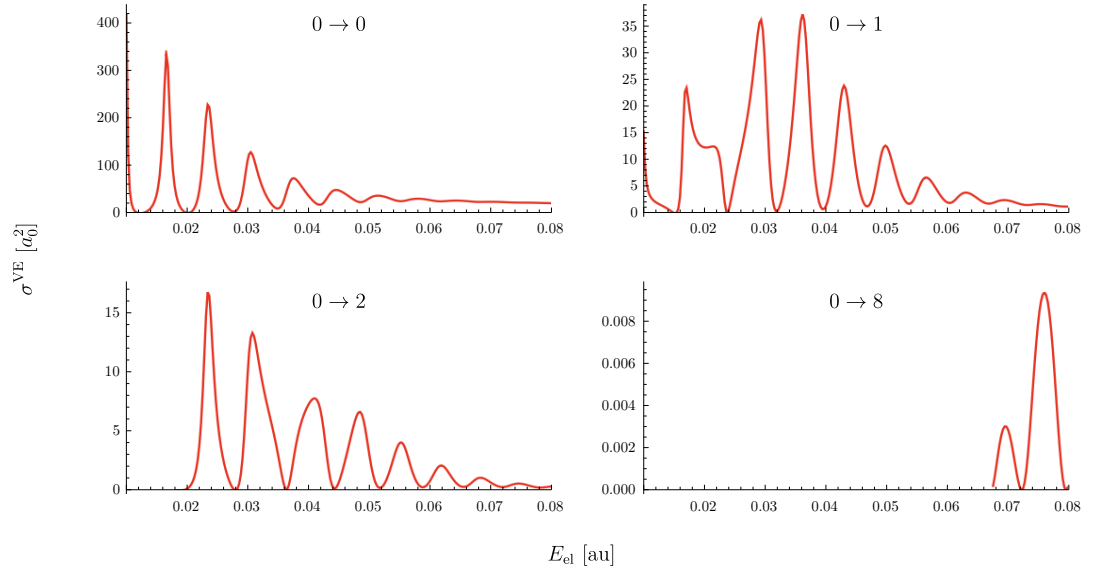


Figure 3.19 The cross-section $\sigma^{\text{VE}}(E_{\text{el}})$ dependence on initial electron energy E_{el} obtained for the diatomic NO via the ILU0 preconditioned iterative method ($\xi_t = 10^{-8}$), 4 different transitions and Gauss-Lobatto quadrature of order $(G_m, G_e) = (10, 8)$.

O₂

$$(G_m, G_e) = (10, 8)$$

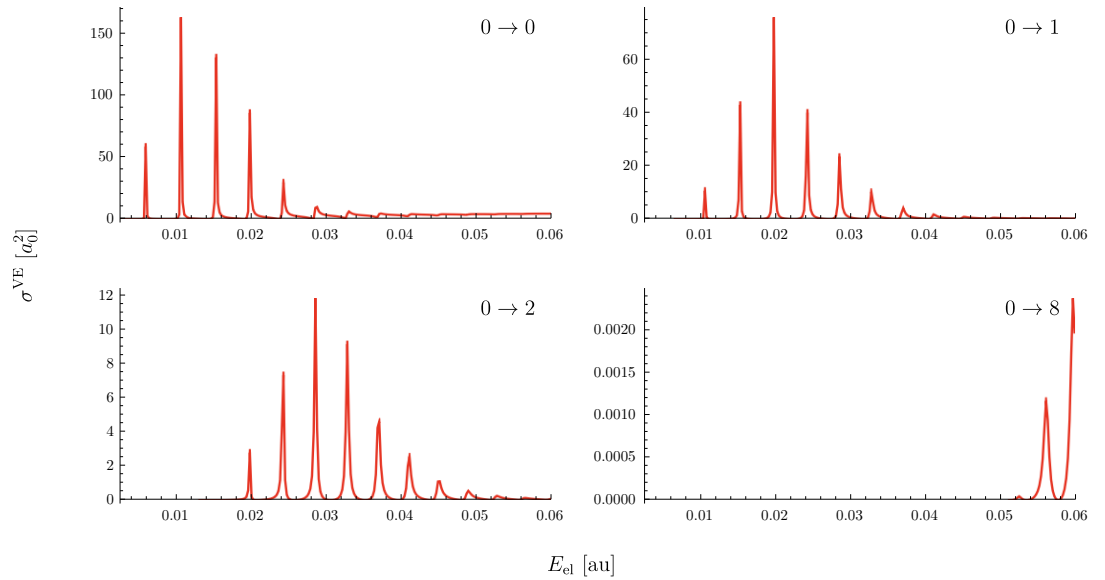


Figure 3.20 The cross-section $\sigma^{\text{VE}}(E_{\text{el}})$ dependence on initial electron energy E_{el} obtained for the diatomic O₂ via the ILU0 preconditioned iterative method ($\xi_t = 10^{-8}$), 4 different transitions and Gauss-Lobatto quadrature of order $(G_m, G_e) = (10, 8)$.

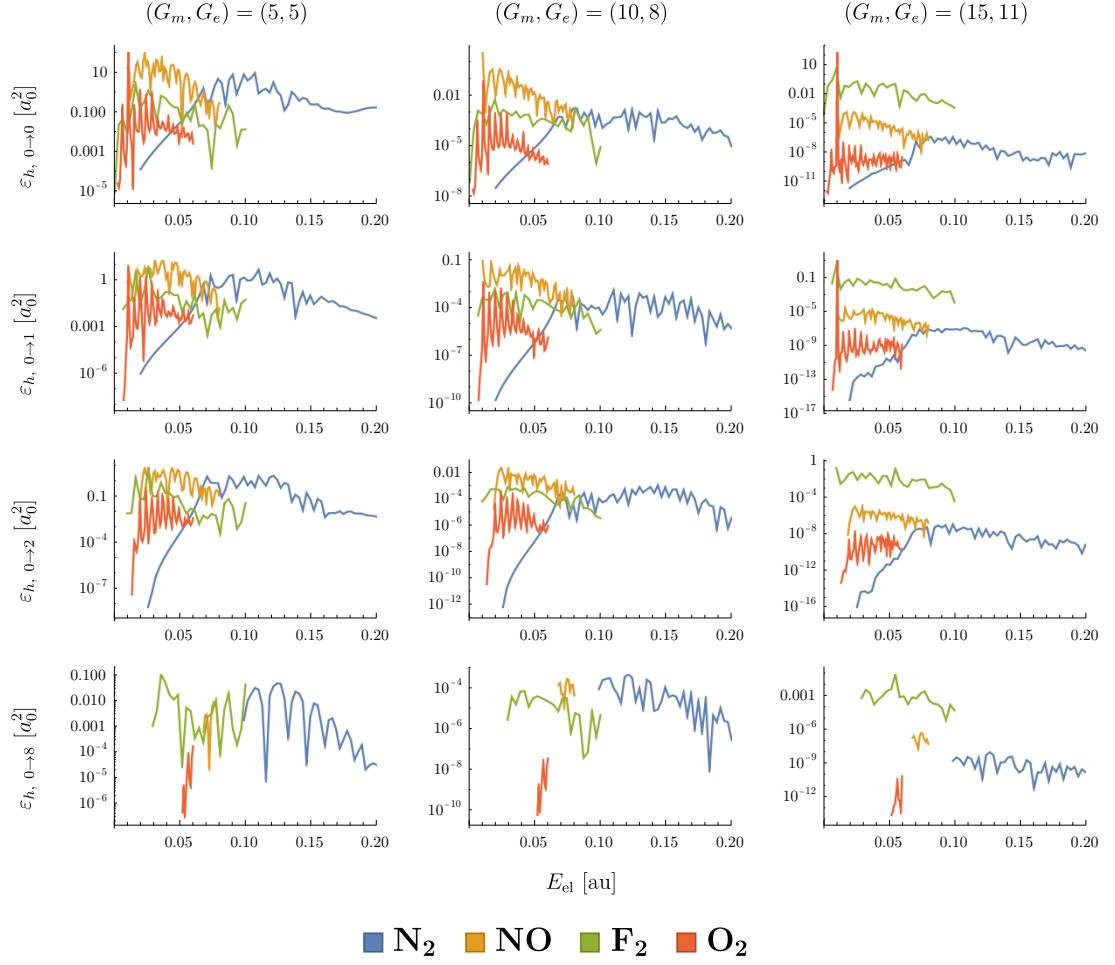


Figure 3.21 The absolute difference $\varepsilon_{h, i \rightarrow f}$ between the solution σ^{VE} for the diatomic N_2 obtained using the highest order $(G_m, G_e) = (17, 15)$ computed by the direct method and the lower orders (horizontal axis) computed with the ILU0 preconditioner for different transitions (vertical axis).

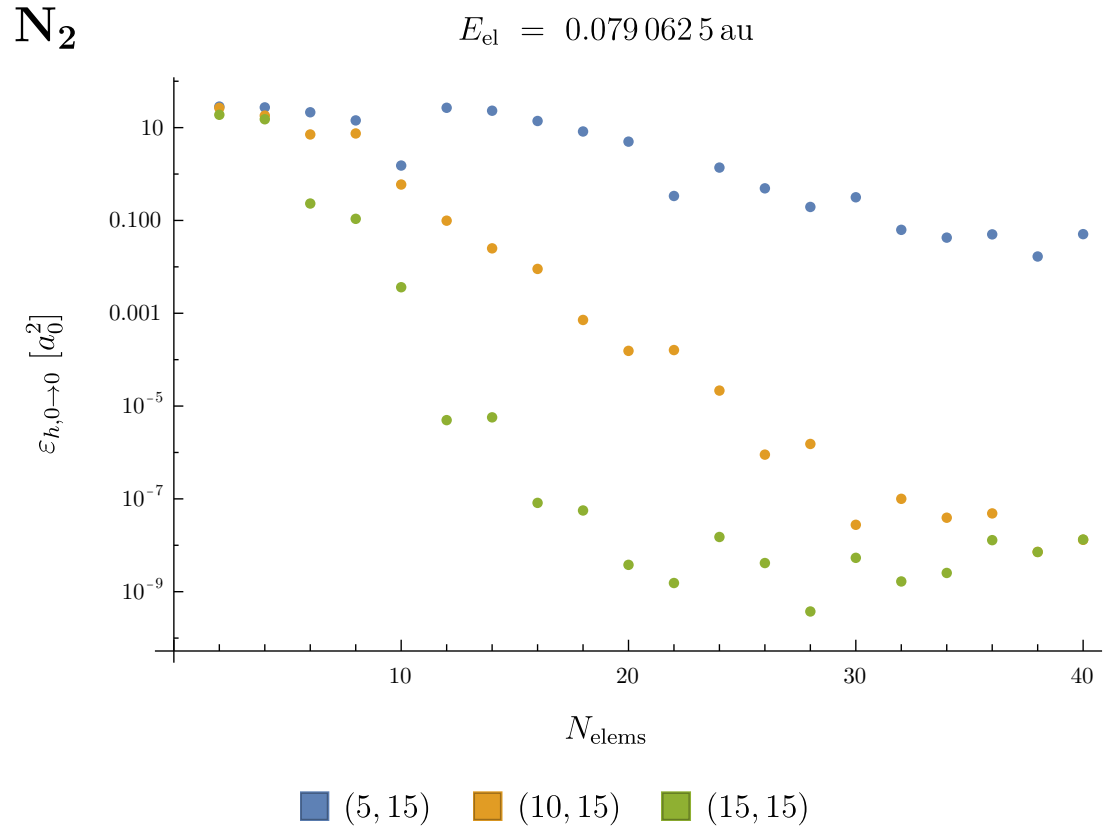


Figure 3.22 The dependence of absolute difference $\varepsilon_{h,0 \rightarrow 0}$ between the solution σ^{VE} obtained using the highest order $(G_m, G_e) = (17, 15)$ and the lower orders on the number of molecular grid points N_m .

F₂

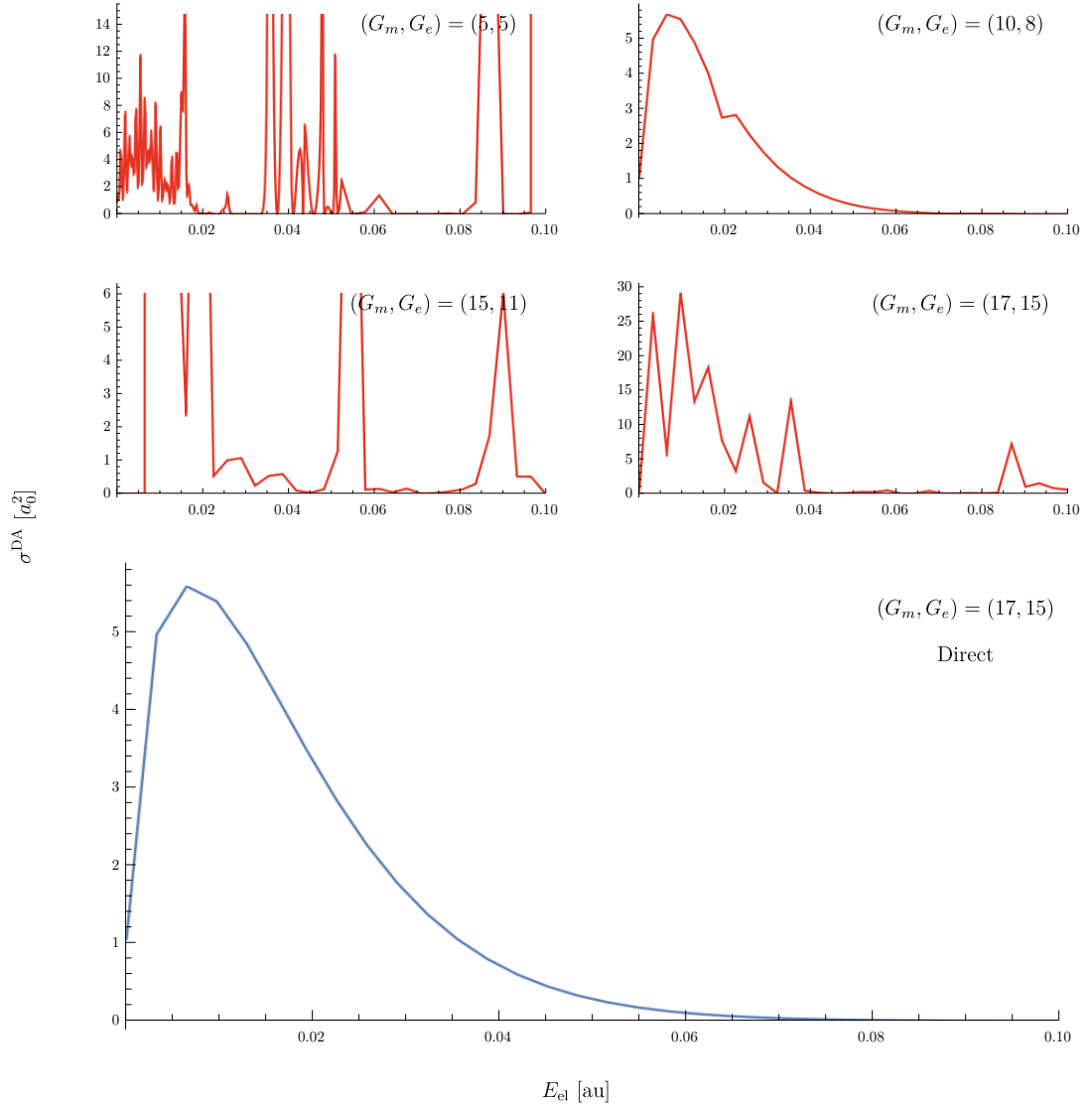


Figure 3.23 The cross-section $\sigma^{\text{DA}}_{\nu_i=0}(E_{\text{el}})$ dependence on initial electron energy E_{el} obtained for the diatomic F_2 via the ILU0 preconditioned iterative method ($\xi_t = 10^{-8}$), 4 different Gauss-Lobatto quadrature orders and the result obtained for the highest quadrature via the direct method. The shape of the curve obtained by the iterative method is completely destroyed by the low order of the quadrature for $(G_m, G_e) = (5, 5)$ and $E_{\text{el}} < 0.05$ and insufficient precision caused by the iteration limit elsewhere. The best iterative result is clearly for the quadrature $(10, 8)$, where the iteration limit of 65000 iterations was enough to get correct qualitative results.

Conclusion

A one-dimensional two-channel model and an existing two-dimensional diatomic model of N_2 , NO and F_2 were studied, with the latter being extended for the case of O_2 . The finite element method, discrete variable representation and exterior complex scaling were overviewed and presented in formalism suitable for scattering problems. Methods for solving linear systems were introduced and an analysis of the sparsity patterns and memory complexities for all relevant matrices was performed.

An iterative COCR solver was then implemented using the Rust programming language and was used with the FEM-DVR-ECS method to solve the presented models. The two-channel model was implemented in a way that allows it to be easily generalized to higher dimensions. Total runtime, number of iterations and convergence properties were studied for an unpreconditioned iterative method, the ILU0 preconditioner and the constructed partial inverse preconditioner. These iterative methods were then compared to the direct zsysv method included in LAPACK and the Intel Direct Sparse Solver. When it comes to performance, the results were as expected: the direct methods are much faster, but also require significantly more memory.

The partial inverse preconditioner was found to be preconditioning the system correctly, although the efficiency was much lower than the ILU0 preconditioner, beating it in the iteration count only in a few specific cases for N_2 . Since partial inverse increases the amount of non-zero elements and is itself expensive to compute, the runtime remained worse than ILU0 even in these cases. It also became evident, that the ordering of the two-dimensional grid plays a major role in the efficiency of this preconditioner and the em ordering, where the inverted blocks take the form of the electronic stiffness matrix, is preferable.

The ILU0 preconditioner was observed as the best option for the majority of covered cases. In the one-dimensional case, its runtime was comparable to the zsysv direct method. In the two-dimensional case however, the Intel DSS method was orders of magnitude faster and the ILU0 preconditioned failed to obtain accurate results under a preset iteration limit for the most complex system F_2 . Although faster, this preconditioner also brought a minor pathological behavior to the convergence of the problem. In some ranges of initial electronic energy, the ILU0 preconditioned iterative method failed to converge either for both orderings, or just the me ordering. These regions seem to be narrower for the em ordering and for this reason such ordering is also preferable with ILU0.

For higher energies and more importantly near the energies where the dissociative attachment channel opens, the studied iterative methods converge much slower and fail to provide accurate results in a reasonable time. For this reason they are better suited for the computation of the vibrational excitation cross-

section.

What remains to be seen next is how these methods deal with the higher-dimensional models, where direct methods cannot be utilized because of the memory constraints. For more demanding systems like F_2 it is already evident, that a more sophisticated preconditioner will be required, whereas for the simpler cases the ILU0 preconditioner could be sufficient.

Bibliography

- [1] John Robert Taylor. *Scattering theory*. New York, London, Sydney, Toronto. John Wiley & Sons, Inc., 1972. ISBN: 0-471-84900-6.
- [2] Roger G Newton. *Scattering theory of waves and particles: Second edition*. en. 2nd ed. Exact solution of the two-channel problem in section 17.3.1. Mineola, NY: Dover Publications, 2003.
- [3] George B. Arfken, Hans J. Weber, and Frank E. Harris. “Chapter 14 - Bessel Functions”. In: *Mathematical Methods for Physicists (Seventh Edition)*. Ed. by George B. Arfken, Hans J. Weber, and Frank E. Harris. Seventh Edition. Boston: Academic Press, 2013, pp. 643–713. ISBN: 978-0-12-384654-9. DOI: <https://doi.org/10.1016/B978-0-12-384654-9.00014-1>. URL: <https://www.sciencedirect.com/science/article/pii/B9780123846549000141>.
- [4] Karel Houfek, T. N. Rescigno, and C. W. McCurdy. “Probing the nonlocal approximation to resonant collisions of electrons with diatomic molecules”. In: *Phys. Rev. A* 77 (1 2008), p. 012710. DOI: 10.1103/PhysRevA.77.012710. URL: <https://link.aps.org/doi/10.1103/PhysRevA.77.012710>.
- [5] Václav Alt and Karel Houfek. “Resonant collisions of electrons with O₂ via the lowest-lying ²Π_g state of O₂[−]”. In: *Phys. Rev. A* 103 (3 2021), p. 032829. DOI: 10.1103/PhysRevA.103.032829. URL: <https://link.aps.org/doi/10.1103/PhysRevA.103.032829>.
- [6] Václav Alt and Karel Houfek. personal communication.
- [7] T. N. Rescigno and C. W. McCurdy. “Numerical grid methods for quantum-mechanical scattering problems”. In: *Phys. Rev. A* 62 (3 2000), p. 032706. DOI: 10.1103/PhysRevA.62.032706. URL: <https://link.aps.org/doi/10.1103/PhysRevA.62.032706>.
- [8] E. Anderson et al. *LAPACK Users’ Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).
- [9] Intel. *Direct sparse solver (DSS)*. URL: <https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top/sparse-solver-routines/direct-sparse-solver-dss-interface-routines.html>.

- [10] Tomohiro Sogabe and Shao-Liang Zhang. “A COCR method for solving complex symmetric linear systems”. In: *Journal of Computational and Applied Mathematics* 199.2 (2007). Special Issue on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2004), pp. 297–303. ISSN: 0377-0427. DOI: <https://doi.org/10.1016/j.cam.2005.07.032>. URL: <https://www.sciencedirect.com/science/article/pii/S0377042705007648>.
- [11] Enrico Bertolazzi and Marco Frego. “Preconditioning Complex Symmetric Linear Systems”. In: *Mathematical Problems in Engineering* 2015 (Mar. 2015). DOI: 10.1155/2015/548609.
- [12] *Richard Ivánek / master-iterative, GitLab*. URL: <https://gitlab.com/RisaI/master-iterative>.
- [13] *GNU time - GNU Project - Free Software Foundation*. URL: <https://www.gnu.org/software/time/>.
- [14] Roger S. Martin and J. H. Wilkinson. “The implicitQL algorithm”. In: *Numerische Mathematik* 12.5 (1968), pp. 377–383. DOI: 10.1007/bf02161360. URL: <https://doi.org/10.1007%2Fbf02161360>.
- [15] *Gfortran - the GNU Fortran compiler, part of GCC*. URL: <https://gcc.gnu.org/wiki/GFortran>.
- [16] *The ‘bindgen’ user guide*. URL: <https://rust-lang.github.io/rust-bindgen/>.
- [17] URL: <https://gcc.gnu.org/onlinedocs/gfortran/Interoperability-with-C.html>.
- [18] Ivan Ukhov et al. *Lapack - rust math library*. 2021. URL: <https://lib.rs/crates/lapack>.

Appendix A

Used grids

Here, we list the grids utilized in each model. We use both the absolute notation (without a sign, simply listing points in the grid) and the additive notation (with a plus sign prefix, listing the differences from the previous point) for the one-dimensional case, but for brevity only the additive notation in the two-dimensional case. In the additive notation, we also use a special notation for equidistant element splitting in the form of $+l/N$, where the element of length l is split into N elements. For example $+6/3$ adds three elements of length 2.

For the two-channel model, the grid in table A.1 was used. For the 2D model, the molecular grids are shown in the table A.2 and electronic in the table A.3. All the grids except for the molecular F_2 grid start from 0.

Absolute $[a_0]$	Additive $[a_0]$
1	$+2/2$
2	
$2 + e^{i\theta}$	$+e^{i\theta}$
$2 + 4e^{i\theta}$	$+3e^{i\theta}$
$2 + 12e^{i\theta}$	$+8e^{i\theta}$
$2 + 28e^{i\theta}$	$+16e^{i\theta}$

Table A.1 The grid used for the two-channel one-dimensional model, $\theta = \frac{\pi}{4}$.

N_2 Additive $[a_0]$	NO Additive $[a_0]$	F_2 Additive $[a_0]$	O_2 Additive $[a_0]$
+1	+1.5	2 (starting point)	+1.5/2
+0.5	+0.5	$+8/80$	+3.5/35
+1.5/6	+2/8	$+15e^{i\theta}/40$	+10/10
+1	+1.5/3		
+2	+1		
$+5e^{i\theta}$	+2		
$+15e^{i\theta}$	$+5e^{i\theta}$		
$+80e^{i\theta}/4$	$+15e^{i\theta}$		
	$+20e^{i\theta}/4$		

Table A.2 The molecular grids for the two-dimensional model, $\theta = \frac{\pi}{4}$.

N ₂ Additive [a_0]	NO Additive [a_0]	F ₂ Additive [a_0]	O ₂ Additive [a_0]
+1	+1	+1.5/3	+1.5/6
+2	+2	+1	+0.5
+3	+3	+2	+1
+4	+4	+60/8	+3
+40/8	+40/8	+5 $e^{i\theta}$	+65/13
+5 $e^{i\theta}$	+5 $e^{i\theta}$	+15 $e^{i\theta}$	+5 $e^{i\theta}$
+15 $e^{i\theta}$	+15 $e^{i\theta}$	+80 $e^{i\theta}$ 4	+15 $e^{i\theta}$
+80 $e^{i\theta}$ /4	+80 $e^{i\theta}$ /4		+80 $e^{i\theta}$ /4
			+100 $e^{i\theta}$ /2

Table A.3 The electronic grids for the two-dimensional model, $\theta = \frac{2}{9}\pi = 40^\circ$.

Appendix B

Implementation details

This appendix contains a few notes regarding the choice of the programming language and some commented samples from the source code of the solver (available in [12]). More importantly, the section B.3 shows how the interface of the product grid looks and how it's built to handle an arbitrary number of dimensions, as was one of the optional assignments of this work.

B.1 Rust Programming Language

The Rust programming language was chosen for its memory safety, performance, convenient and reliable tooling and many high level language features it provides. Since the previous work in this field was made primarily in FORTRAN, one of the first goals was to test how difficult it would be to get FORTRAN code to interop with new Rust code. As it turns out, this was not difficult at all.

Three FORTRAN source files (used for calculating the Gauss-Lobatto quadrature points and weights[14] and the derivatives of Lagrange polynomials) were placed in the *fortran/* directory inside the codebase. Next, the build script *build.rs* was written which does the following things:

1. Finds all FORTRAN 77 files with the extension *.f* and compiles them using *gfortran*[15] to produce *.o* files.
2. Finds all FORTRAN 90 files with the extension *.f90* and compiles them using *gfortran* with the 2003 revision of the language to produce *.o* files.
3. Links the produced *.o* files to produce a binary *libgeneral.a* file and stores it in the *target/* directory.
4. Generates a *general.h* C header file using *gfortran* which contains all ISO C function signatures from the detected *.f90* files.
5. Uses the *rust-bindgen*[16] tool to generate Rust bindings for the previously generated *general.h* header file and stores them in the *general.rs* file inside the target artifacts.
6. Instructs *rustc* to link the produced binaries and to statically link the *gfortran* lib.

Next, inside the source code, the generated *general.rs* file is included and this makes all the ISO C functions available to Rust code. For more information on how to enable C interop in FORTRAN code, refer to [17].

Next, a LAPACK implementation was integrated into the codebase. There is already an existing *lapack* crate[18] that enables this by just importing it and selecting the specific LAPACK implementation as a crate feature (*intel-mkl* for our case).

Lastly, the Intel Direct Sparse Solver had to be integrated. There is currently no crate to streamline this, but since the *lapack* already links Intel MKL, the only thing that had to be done was to generate unsafe DSS bindings using *rust-bindgen*.

The entire procedure was straightforward and this type of FORTRAN interop could easily be generalized to the Intel FORTRAN compiler and could be distributed as a build time crate for future work.

B.2 Constructing grids

To enable arbitrary grid construction, a generic *Grid<X>* type was implemented. It has the following interface:

```
pub struct Grid<X> {
    verts: Vec<X>,
}

impl<X> Grid<X> {
    // Create an empty grid
    pub fn empty() -> Self;
    // Check if the grid is empty
    pub fn is_empty(&self) -> bool;
    // Add a vertex without checking for order
    pub fn add_vert_unchecked(&mut self, point: impl Into<X>) ->
        &mut Self;
}

impl<X> Grid<X>
where
    X: ComplexField + Copy,
{
    // Append 'segments' points of total length 'len' to the grid
    pub fn append_equidist(&mut self, len: impl Into<X>, segments
        : usize) -> &mut Self;
    // Append elements of specified lengths
    // ex.: grid.append_many([1., 2., 3.]) to append 3 elements
    // of total length 6
    pub fn append_many<T: Into<X>>(&mut self, dists: impl
        IntoIterator<Item = T>) -> &mut Self;
    // Create the final grid by selecting a quadrature
    pub fn into_quadrature_grid(self, quad: Quadrature) ->
        QuadratureGrid<X, X>;
}
```

The *Quadrature* type is a great example for the use of discriminated unions (also simply called Rust enums):

```
pub enum Quadrature {
    // Trapezoidal rule
    Simple,
    // Gauss-Lobatto quadrature of a given order
    GaussLobatto(usize),
}
```

Then, the final *QuadratureGrid*<*X*, *W*> type contains the grid points and weights and can be used to build the stiffness matrix and represent functions as vectors:

```
pub struct QuadratureGrid<X, W> {
    // Only informational
    quadrature: Quadrature,
    original_points: Box<[X]>,
    // These are necessary
    points: Box<[X]>,
    weights: Box<[W]>,
}

impl<X, W> QuadratureGrid<X, W> {
    pub fn get_quadrature(&self) -> &Quadrature;
    pub fn get_point(&self, idx: usize) -> &X;
    pub fn get_weight(&self, idx: usize) -> &W;

    pub fn iter_points(&self) -> impl DoubleEndedIterator<Item =
        &X>;
    pub fn iter_weights(&self) -> impl Iterator<Item = &W>;
    pub fn iter_points_weights(&self) -> impl Iterator<Item = (&X
        , &W)>;

    pub fn represent_fn_dvr<F, Y>(&self, f: F) -> Vec<Y>
    where
        F: Fn(X) -> Y,
        X: Clone,
    {
        self.points.iter().map(|x| f(x.clone())).collect()
    }
}

impl<T> QuadratureGrid<T, T>
where
    T: ComplexField + Clone,
{
    pub fn represent_fn<F: Fn(T) -> T>(&self, f: F) -> Vec<T> {
        self.points
            .iter()
            .zip(self.weights.iter())
            .map(|(x, w)| f(x.clone()) * w.clone().sqrt())
            .collect()
    }

    pub fn integrate_fn<F: Fn(T) -> T>(&self, f: F) -> T;
    pub fn stiffness_matrix_whole(&self, prefactor: T) ->
        nalgebra_sparse::CooMatrix<T>;
}
```

The grid for the one-dimensional model is constructed like in this snippet:

```
pub fn build_grid(
    gl_order: usize,
    theta: f64,
) -> QuadratureGrid<C64, C64> {
    let mut grid = Grid::::empty();
    let exp = C64::new(0., PI * theta / 180.).exp();

    grid.append_equidist(2., 2) // 0-1-2
        .append_many([1., 3., 8., 16.].map(|v| v * exp));

    grid.into_quadrature_grid(Quadrature::GaussLobatto(gl_order))
}
```

B.3 The product grid

To construct a multidimensional grid, you first generate a *QuadratureGrid* for each coordinate. Then, you use the *ProductGrid* type to combine the *QuadratureGrids* into a single multidimensional grid. The definition of this type uses constant generic to enable an arbitrary amount of dimensions while still being able to do many optimizations at build time and also to enable strict type checking along the way. The interface of this struct looks like this:

```
pub struct ProductGrid<X, W, B: Borrow<QuadratureGrid<X, W>>,
    const DIM: usize> {
    grids: [B; DIM],
    ..
}

impl<X, W, B: Borrow<QuadratureGrid<X, W>>, const DIM: usize>
    ProductGrid<X, W, B, DIM> {
    pub fn new(grids: [B; DIM]) -> Self {
        Self {
            grids,
            ..
        }
    }

    pub fn get_point(&self, idx: [usize; DIM]) -> [X; DIM];
    pub fn get_weight(&self, idx: [usize; DIM]) -> W;
    pub fn get_index_iterator(&self) -> MultiIndexIterator<DIM>;

    pub fn represent_fn_dvr<F, Y>(&self, f: F, borderless: bool)
        -> Vec<Y>
    where
        X: Clone,
        F: Fn([X; DIM]) -> Y;

    pub fn represent_fn<F>(&self, f: F, borderless: bool) -> Vec<
        W>
    where
        X: Clone,
        W: ComplexField + Clone + Product,
        F: Fn([X; DIM]) -> W;
```

```

    pub fn integrate_fn<F>(&self, f: F) -> W
    where
        X: Clone,
        W: ComplexField + Clone + Sum + Product,
        F: Fn([X; DIM]) -> W;
}

impl<T, B: Borrow<QuadratureGrid<T, T>>, const DIM: usize>
    ProductGrid<T, T, B, DIM> {
    pub fn stiffness_matrix<P>(&self,
        prefactors: [P; DIM],
        borderless: bool,
    ) -> nalgebra_sparse::CooMatrix<T>
    where
        P: Into<T> + Copy,
        T: ComplexField + Sum + Copy;
}

impl<T, B: Borrow<QuadratureGrid<T, T>>> ProductGrid<T, T, B, 2>
{
    pub fn stiffness_matrix_fast<P>(&self,
        prefactors: [P; 2],
        borderless: bool,
    ) -> nalgebra_sparse::CooMatrix<T>
    where
        P: Into<T> + Copy,
        T: ComplexField + Sum + Copy;
}

```

The multidimensional functions are one parameter functions of a statically sized array of length DIM . A general function for generating the product stiffness matrix is implemented in the *stiffness_matrix* method. A much faster implementation for the specific case of $DIM = 2$ is implemented in the *stiffness_matrix_fast* method. The fact that we can implement functions conditionally based on restrictions of the generic types and even the constant generic type DIM ensures that the function will be available if and only if the dimension is 2 and the numeric type is capable of representing the Gauss-Lobatto points and weights.

The constant generic types are a very young feature of the Rust programming language. In the future, it could be used for example to program a tensor library, where the dimension and the numbers of covariant and contravariant indices are known at compile time and can be type checked to eliminate errors while writing the code. After constant generic expressions are stabilized, the library would be able to automatically change the number of indices when contracting or doing tensor multiplication.

Having implemented the *ProductGrid* type in this generic way, the final assignment of this master thesis is complete, as the program is ready to be used on higher dimensional models and even with different numeric types for which the necessary operations are implemented. For more comments, details and the implementations themselves, please refer to the source code[12].

