ARCHER eCSE Final Report

POpSiCLE: A photoelectron spectrum library for laser-matter interactions

Daniel Dundas & Alejandro de la Calle
Atomistic Simulation Centre
School of Mathematics and Physics
Queen's University Belfast

6 June 2016

Abstract

In this project we have developed a parallel library that implements three methods for calculating photoelectron spectra. These are the surface flux method, the sampling point method and the Fourier transform of the spatial wavefunction into momentum space. The library has been written to be as portable as possible and can be interfaced straightforwardly with other codes that are used for modelling laser-molecule interactions. These include several codes written by the applicants that solve the time-dependent Schrödinger equation (TDSE) for simple molecules (THeREMIN and RHYthMIC) and solve the Kohn-Sham equations of time-dependent density functional theory (TDDFT) for complex molecules (EDAMAME). In addition other TDDFT community codes can also easily use the library.

Contents

1 Introduction

Molecules driven out of equilibrium by intense, ultrashort laser pulses are of central importance in many areas of science and technology. In such a non-equilibrium situation, charge and energy transfer can be induced across the molecule on a femtosecond timescale. Understanding and controlling these transfer processes is fundamental to many chemical processes and key to future ultrafast technologies: examples include the design of electronic devices, probes and sensors, biological repair and signalling processes and development of optically driven ultrafast electronics.

During the interaction of these pulses with molecules a range of processes occur such as ionization, dissociation and harmonic generation. In order to understand and control the interactions, we must be able to analyse the photo-fragments produced, i.e. the electrons, ions and photons. Being able to calculate the photoelectron energy spectrum (PES) of ionizing electrons provides an extremely sensitive tool with which we can study the electronic structure of the molecule at the point of ionization or dissociation. However, calculating the PES is notoriously difficult. This is because the spectrum should be calculated as the projection of the electronic wavepacket onto continuum states at the end of the pulse. This presents a real challenge as electrons ionized during the pulse attain high velocities and can readily travel to extremely large distances in a short time. This makes the calculation prohibitively expensive since the wavepacket must be calculated on a large numerical grid. In general, approximate methods are used whereby the electron energies are calculated during the pulse as they pass detector points or measuring surfaces before being removed from the simulation volume.

In describing laser-molecule interactions a large number of community codes are used. Each code generally provides it's own implementation of an approach for calculating photoelectron spectra (or in some cases no implementation is available). This means that much effort goes into implementing the same approach many times. In this project we have developed a parallel library of routines for calculating the PES using three different approaches. These approaches are

1. the Fourier transform of the spatial wavefunction into momentum space,
2. the surface flux method, and
3. the sampling point method.

The library has been written to be as portable as possible and can be interfaced straightforwardly with other codes that are used for modelling laser-molecule interactions. These include several codes written by the applicants that solve the time-dependent Schrödinger equation (TDSE) for simple molecules (THeREMIN and RHYthMIC) and solve the Kohn-Sham equations of time-dependent density functional theory (TDDFT) for complex molecules (EDAMAME).

In this document we describe the implementation of the library. In Section 2 we describe the three methods that are implemented in the library. Section 3 then describes how each of these methods is implemented in turn in our library. Some illustrative results are presented in Section 4. Finally, in Section 5 some conclusions are drawn and details for obtaining library, POpSiCLE (PhOtoelectron SpeCtrum Library for lasEr-matter interactions), are given.

2 Theoretical methods for calculating photoelectron spectra

In the following section we will briefly describe the three methods that are implemented in POpSiCLE for calculating photoelectron spectra (PES). In section 2.1, we show how we can obtain

the PES, together with photoangular distributions (PADs) using the Fourier transform method. In addition, the calculation of joint electron-ion spectra for the hydrogen molecular ion (H_2^+) using the Fourier method will be described. In section 2.2 we review the time-dependent surface flux method (t-SURFF) for single electron ionisation and joint electron-ion spectra. Lastly, in section 2.3 the sampling point method is described for calculating both PES and PADs.

2.1 The Fourier method

The simplest method to extract asymptotic information about outgoing electron energies is to project the wavefunction onto scattering solutions, $\psi_k(\boldsymbol{r})$, that describe time-independent dynamics after the field interaction. These scattering solutions obeys the TDSE

$$H(T)\,|\psi_k\rangle = \frac{\boldsymbol{k}^2}{2}\,|\psi_k\rangle\,, \tag{1}$$

where $H(T)$ is the field-free Hamiltonian at a long time T after the interaction, $\boldsymbol{k}$ is the wave vector and k is the wavenumber. To do this, we can think of coordinate space divided in two regions: an inner region which contains all the bound states of the system, and an outer region which contains the scattering products of the interaction. Both regions are separated by a boundary surface at radius r_b. Therefore, for sufficiently long times after the interaction, we can split the wavefunction as

$$\Psi(\boldsymbol{r}, T) = \Psi_b(\boldsymbol{r}, T) + \Psi_s(\boldsymbol{r}, T) \tag{2}$$

where $\Psi_b(\boldsymbol{r}, T)$ is the bound wavepacket, $\Psi_s(\boldsymbol{r}, T)$ is the free wavepacket and T is large enough that the asymptotic part has time to cross the boundary surface r_b. In this case

$$\Psi_b(\boldsymbol{r}, T) \approx 0 \qquad \text{for} \quad |\boldsymbol{r}| \geq r_b \tag{3}$$

$$\Psi_s(\boldsymbol{r}, T) \equiv \int d\boldsymbol{k}\; e^{-iT\boldsymbol{k}^2/2} b(\boldsymbol{k}) \psi_k(\boldsymbol{r}) \approx 0 \qquad \text{for} \quad |\boldsymbol{r}| \leq r_b \tag{4}$$

where $b(\boldsymbol{k})$ are the spectral amplitudes, which can be obtained by spectral decomposition of the wavefunction

$$b(\boldsymbol{k}) = \langle \psi_k | \Psi_s(T) \rangle\; e^{-iT\boldsymbol{k}^2/2}. \tag{5}$$

Photoionisation implicitly assumes that, at large distances from the interaction region (where the detectors are placed), and at large times after the interaction, the scattering solutions can be described as free particles, i.e. plane waves and so we can write

$$\psi_k(\boldsymbol{r}) \approx \chi_k(\boldsymbol{r}) = \frac{1}{\left(\sqrt{2\pi}\right)^3}\; e^{i\boldsymbol{k}\cdot\boldsymbol{r}}. \tag{6}$$

Now, we can decompose the spectral amplitudes by plane waves

$$b(\boldsymbol{k})\; e^{iT\boldsymbol{k}^2/2} = \langle \chi_k | \Psi_s(T) \rangle = \int d\boldsymbol{r}\; \chi_k^*(\boldsymbol{r}) \Psi_s(\boldsymbol{r}, T) = \frac{1}{\left(\sqrt{2\pi}\right)^3} \int d\boldsymbol{r}\; e^{-i\boldsymbol{k}\cdot\boldsymbol{r}} \Psi_s(\boldsymbol{r}, T). \tag{7}$$

The right-hand side of Eq. (7) is merely the Fourier transform of the scattering part of the wavefunction [1, 2]. The only requirement imposed by the Fourier transform is that the canonical momentum of the wavefunction must be the same as the mechanical momentum. This means that in a velocity gauge description of the interaction, the calculation must be done after the pulse has finished (when the vector potential and its first two derivatives are zero).

2.2 The t-SURFF method

The time-dependent surface flux method (t-SURFF) has been developed by Tao and Scrinzi for one and two electron systems [3, 4], and it has been further applied to the strong field ionisation of H_2^+ [5, 6], and more complex systems [7]. In this method we again divide configuration space into two regions, as given in Eq. (2). Rather than storing the scattering solution and Fourier transforming this, we record the outgoing flux of electrons passing through the surface between the two regions. In the outer region we assume the wavepackets evolve as Volkov waves, i.e. the dynamics are only governed by the electromagnetic field. This assumption allows to convert the volume integral, Eq.(7), into a time integral of a time-dependent flux passing through a surface. The fact that the outgoing wavepackets are only stored at certain points on the grid allows the use of absorbing boundaries at the edge of the boxes, reducing dramatically the grid sizes needed.

The spectral amplitudes will be expressed as in Eq. (7), i.e.

$$b(\boldsymbol{k})\, e^{iT\boldsymbol{k}^2/2} = \langle \chi_k | \Psi_s(T) \rangle = \langle \chi_k |\, \theta(r_b)\, | \Psi_s(T) \rangle \equiv \int_{|\boldsymbol{r}|>r_b} d\boldsymbol{r}\; \chi_k^*(\boldsymbol{r}) \Psi_s(\boldsymbol{r}, T). \tag{8}$$

The key of the method is to convert the volume integral on the right-hand-side of Eq. (8) into a time-dependent surface integral. In order to do that, we must know the evolution of the wavefunction after it has passed the surface. The Hamiltonian which rules the dynamics in the outer region is the Volkov Hamiltonian, which has the form

$$\mathcal{H}_V(t) = \frac{1}{2}\; [\boldsymbol{k} - \boldsymbol{A}(t)]^2 \tag{9}$$

The solutions to the correspondent TDSE are the so-called Volkov solutions

$$\chi_k(\boldsymbol{r}) = \frac{1}{\left(\sqrt{2\pi}\right)^3}\; e^{i\boldsymbol{k}\boldsymbol{r}} e^{-i\Phi(\boldsymbol{k},t)} \tag{10}$$

where

$$\Phi(\boldsymbol{k}, t) = \frac{1}{2} \int_0^t dt'\; \left[\boldsymbol{k} - \boldsymbol{A}(t')\right]^2 \tag{11}$$

is the Volkov phase and $\boldsymbol{A}(t)$ is the vector potential of the electric field.

In order to transform the volume integral in Eq. (8) to the time integral of the surface flux we write

$$\begin{aligned} \langle \chi_k |\, \theta(r_b)\, | \Psi_s \rangle &= \int_0^T dt\; \frac{d}{dt} \Big\langle \chi_k \Big| \theta(r_b) \Big| \Psi_s(T) \Big\rangle \\ &= i \int_0^T dt \Big\langle \chi_k \Big| \mathcal{H}_V(t)\theta(r_b) - \theta(r_b)\mathcal{H}(t) \Big| \Psi_s(T) \Big\rangle \\ &= i \int_0^T dt \left\langle \chi_k \left| \left[-\frac{1}{2}\nabla^2 - i\boldsymbol{A}(t)\cdot\boldsymbol{\nabla}, \theta(r_b) \right] \right| \Psi_s(T) \right\rangle. \end{aligned} \tag{12}$$

Yue and Madsen [6] extended this approach in order to calculate the joint electron-ion spectra for H_2^+. The basis of this method is similar to the method outlined above. In this case we require two surfaces, one for the electronic degrees of freedom (r_b above) and a new surface (at R_b) for the dissociation coordinate. In addition to the electronic scattering solutions described above (the Volkov waves) we also need to include ionic scattering solutions (plane waves).

2.3 Sampling point method

This method was originally proposed by Suraud and co- workers to study photo-fragmentation of clusters [8]. In an actual experiment, detectors would be placed at various points to collect the

photoelectrons produced. The sampling point methods uses this idea to calculate the PES. In this case we set up a number of sampling points on a sphere of given radius. The electronic wavepacket passing each point is collected during the simulation and then Fourier transformed at the end. With a large number of sampling points we obtain the angularly-resolved PES. There are several benefits of this method compared with the Fourier method shown previously. Firstly, smaller grids are required for the sampling point method since we only record electron wavepacket passing the sampling points. Secondly, the full Fourier transform of the wavefunction to momentum space (an expensive volume integral) is not required. The method gives the PES which is proportional to the spectral amplitudes by evaluating the energies that crosses the measuring points. It is useful in situations in which we want to know the shape of the PES, or the position of the peaks in the spectra but not when we are interested in the actual values of the scattering amplitudes.

Consider a particular sampling point $\boldsymbol{r}_m$. If we record the wavefunction passing this point during the interaction of the laser pulse, Fourier transforming this gives

$$\Psi_s(\boldsymbol{r}_m, \omega) = \frac{1}{\sqrt{2\pi}} \int dt\ e^{j\omega t} \Psi(\boldsymbol{r}_m, t). \tag{13}$$

We can show that $\Psi_s(\boldsymbol{r}_m, \omega)$ is related with to the spectral amplitude, $b(\boldsymbol{k})$. Then, assuming that at each sampling point $\boldsymbol{k} \parallel \boldsymbol{r}_m$, we have

$$\Psi_s(\boldsymbol{r}_M, \omega) = 4\pi k^2 b(\sqrt{2\omega_k}) e^{ir_m\sqrt{2\omega_k}} \tag{14}$$

where $E = \omega_k = k^2/2$. The PES yield is given simply by

$$\frac{dP_m(E)}{dE} = |b(\boldsymbol{k})|^2 \propto |\Psi_s(\boldsymbol{r}_M, E)|^2, \tag{15}$$

and average of the PES from all the measuring points across a sphere is the total PES

$$\frac{dP(E)}{dE} = \sum_M \frac{dP_M(E)}{dE}\ \gamma_m. \tag{16}$$

In order to average between the points the weighting factors, γ_M, have been introduced in Eq. (16). Each weighting factor is just the solid angle covered by each sampling point.

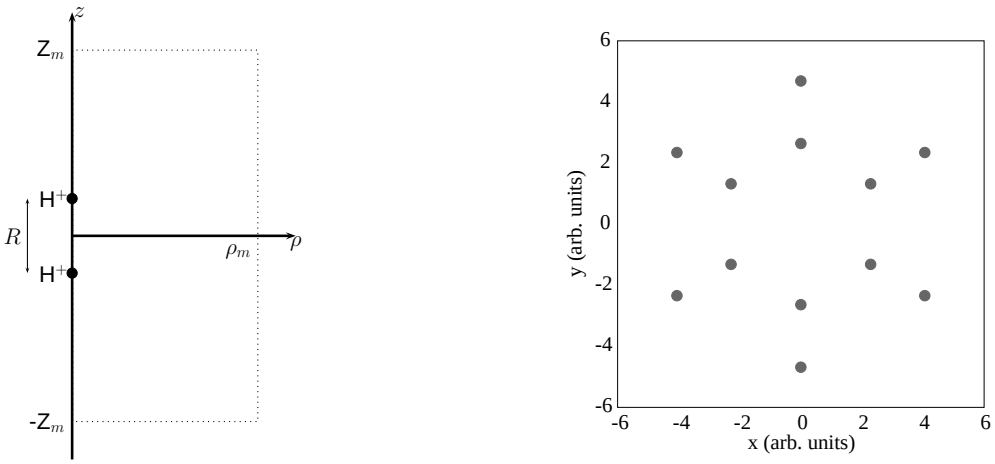
3 Numerical implementation of the library

The methods outlined in the previous section have been implemented for use in two of our codes, THeREMIN and EDAMAME. THeREMIN solves the TDSE for the H_2^+ in cylindrical coordinates. EDAMAME solves the time-dependent Kohn-Sham equations for time-dependent density functional theory (TDDFT) on adaptive finite difference grids in Cartesian coordinates. The spatial regions in a typical simulation for each code is summarised in Figure 1.

We can see that the grid layout is quite different in both cases. Therefore, in order to produce a portable library we have needed to provide a range of tools for handling these different situations. At the same time we want to make use of standard approaches that can be reused where possible.

For the implementation of the Fourier transform method we thus make use of FFTs when dealing with coordinates spanning the range $(-\infty, \infty)$ and spherical Bessel transforms for coordinates spanning the range $(0, \infty)$.

In the case of the surface flux method and the sampling point method, each requires knowledge of the wavefunction on a spherical surface. Therefore, we decided to work in spherical coordinates. In



(a) (b)

Figure 1: Schematic diagram of the spatial grids used in THeREMIN and EDAMAME. In (a) the cylindrical grid used by THeREMIN is shown. In this case if the laser field is aligned along the molecular axis, which is itself aligned along the *z*-axis, the system possesses cylindrical symmetry and so we only need to describe the ρ and *z* coordinates. In (b) the Cartesian grid used in EDAMAME is shown. In this case a full 3D grid is used but only two dimensions are shown for clarity. For a typical simulation the interaction of an arbitrarily polarised laser field with a general polyatomic molecule is considered and so no special symmetry is present in the problem. In the case presented here, the benzene molecule (C_6H_6) is considered.

that case, we provide a set of routines to interpolate the wavefunction on this surface and then provide an implementation of each approach in spherical coordinates.

POpSiCLE was created to be used as an external library, with a range of subroutines that can be called within TDSE/TDDFT solvers. In addition, it contains helper scripts for post-processing and plotting data. The core routines of POpSiCLE are written in FORTRAN 2003, with the post-processing scripts mainly written in FORTRAN 2003 and python. The library can be used for both serial and parallel calculations. One of the main purposes of the library was to create a set of tools that can be used in massively parallelised High-Performance Computing (HPC) codes. For efficient communication between nodes the library uses the MPI protocol while for efficient I/O the HDF5 file format is used.

One additional requirement is a method to calculate FFTs. With suitable compilation flags, the user can choose either the Intel Math Kernel Library (MKL) [9] or the standalone Fastest Fourier Transform in the West (FFTW) [10]. MKL is a proprietary and very efficient library for computing FFTs. It is specially designed to have higher performance on Intel architectures. FFTW is another efficient library, written in C, to compute the FFT algorithm. It available across different architectures and is one of the most popular implementation for FFT calculations in many scientific calculations.

In this section we will describe the numerical implementation of the library. We have divided this section in three parts. In section 3.1 we will discuss the implementation of the Fourier method. Section 3.2 will then describe how we interpolate the wavefunction information onto a spherical surface for use in the t-SURFF and sampling point methods. In section 3.3 the implementation of the t-SURFF methods will be described while the implementation of the sampling point method will be described in section 3.4.

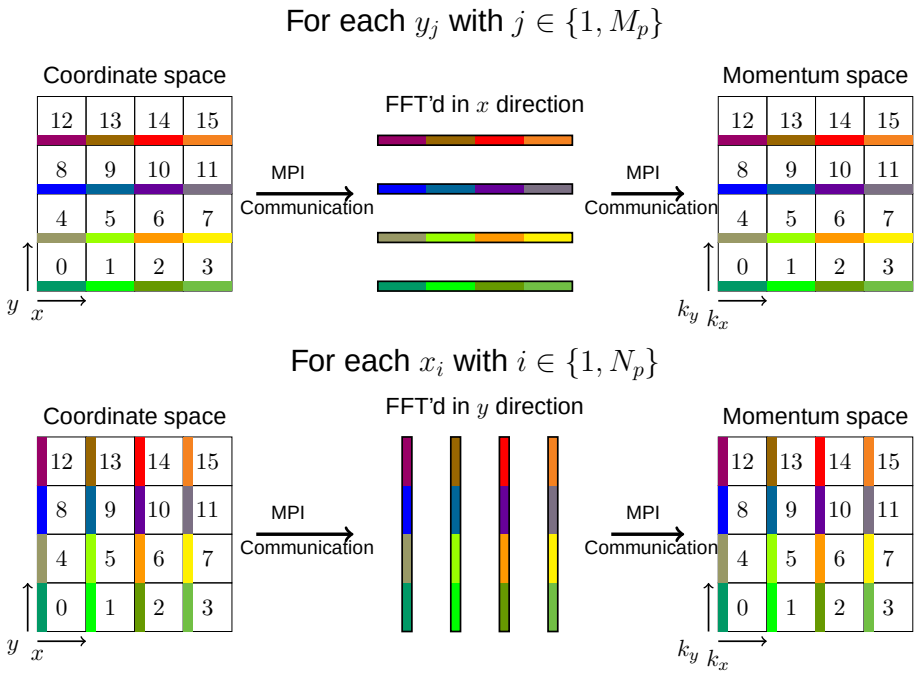


Figure 2: Schematic diagram showing how the FFT of a wavefunction parallelised on a 2D grid can be carried out. See text for a complete description.

3.1 Implementation of the Fourier method

For equidistant coordinates, the most efficient way to transform the wavefunction to momentum space is to use the Fast Fourier Transform (FFT) [11, 12]. The FFT algorithm exploits the symmetry between the odd and even terms of a DFT, following a 'divide and conquer' strategy. This algorithm scales as $O(N \log N)$, which means that for an array of length $N = 10^6$ the calculation will last around 50 ms, whereas with the DFT it will take around 20 hours.

Two of the most popular implementations of the FFT method are the Intel MKL [9] and the open-source FFTW (Fastest Fourier Transform in the West) [10]. Either of these implementations can be chosen at compile time.

In our case, we have to perform the FFT of large multi-dimensional arrays obtained through parallel calculation. This imposes memory constraints and so our strategy is to implement a set of subroutines that redistributes the decomposition of the wavefunction itself. To illustrate this, we will consider the 2D example shown in figure 2. In this case the wavefunction for the system, lying in the $x - y$ plane, is distributed across 16 processes, arranged as a 4×4 grid. Along the x axis, N_p points are stored per process so that the total number of x-points in the calculation is $N = 4N_p$. Along the y axis, M_p points are stored per process so that the total number of y-points in the calculation is $M = 4M_p$. The calculation begins with each process containing its own piece of the total grid. We will FFT the wavefunction for each coordinate in turn. Consider the FFT in the x coordinate. Each process will have to carry out an FFT M_p times (one for each y value). Each FFT requires the wavefunction information for all N points along the x-axis, but each process only has N_p elements. We communicate this particular slice of the wavefunction to all the processes which participate in this direction with a `MPI_Alltoall` call. After the FFT completes each process obtains its own range of the corresponding k_x values (the momentum values in x) with another `MPI_Alltoall` call. Once the FFTs are calculated along the x coordinate, the process is repeated for the y coordinate.

For the case of THeREMIN, which solves the TDSE in cylindrical coordinates, we require the Fourier transform of the ρ coordinate, lying in the range $0 \leq \rho \leq \infty$. In this case, we require the Bessel

transform of the wavefunction, i.e.

$$F(k_\rho) = \int_0^{\rho_{max}} d\rho J_0(k_\rho \rho) f(\rho), \tag{17}$$

where $J_0(k_\rho \rho)$ is the zero-order Bessel function of first kind. Since ρ is also distributed across processes, we parallelise this as illustrated in Figure 2.

POpSiCLE provides two main routines for transforming a wavefunction from coordinate space to momentum space

1. `Fourier_transform`
   To call the main routine, `Fourier_transform`, the user must provide the rank and dimensions of the grid, (apart from the input and output array, of course). If the user wants to use the parallel version, he/she also needs to provide the dimensions of the wavefunction in total and per processor, the global MPI communicator and the position that each processor occupies on the total grid.
2. `make_BesselFourier_Transform`
   To call this routine for use with the ρ coordinate in cylindrical coordinates, the user must specify the same inputs for this coordinate as are used for `Fourier_transform`.

3.2 Interpolation of the wavefunction onto spherical surfaces

For both the t-SURFF and the sampling point method, we need to know the value of the wavefunction (and its first derivative) at a series of points on a spherical shell at a given radius from the origin. In light of the spherical nature of the problem and the fact that the initial implementation of t-SURFF was in spherical coordinates, this is the most appropriate coordinate system to use. However, many codes, including those used in our work, are not written in spherical coordinates and so we require a method for interpolating the wavefunction information onto a spherical shell. Since we are only interested in these values on a defined shell at radius r_b, one way to proceed is to calculate these values during the execution of the TDSE/TDDFT solver, store these quantities to file and then post-process this data at a later time. We find this way more flexible because the computational demand of calculating the time-integral in the t-SURFF method and the FFTs for the sampling point method is small compared to the effort required for the TDSE/TDDFT solver.

In order to interpolate the wavefunction at the required surface, the user can choose between two multi-dimensional interpolation algorithms

1. Shepard's method
   The Shepard interpolation is a method used to interpolate scattered data in two and three dimensions. We use the external package of routines SHEPPACK, written in FORTRAN 90, which contains the implementation of the modified Shepard algorithm [13]. Before interpolating, it performs a gridding operation through all points to locate them in an area or volume (depending if the interpolation is 2D or 3D). This method is computationally expensive, and it can cause the huge slow down in the TDSE/TDDFT solver.
2. Bi/tri-cubic interpolation
   For two-dimensional data we implement bicubic interpolation [11] while for three-dimensional data we use tricubic interpolation [14]. Both methods work by carrying out cubic interpolation on 2D and 3D regions of space using piecewise polynomials. The overall performance of the bi/(tri)-cubic method is a couple of orders of magnitude faster than Shepard's method at the same level of error. For that reason is POpSiCLE's default interpolator.

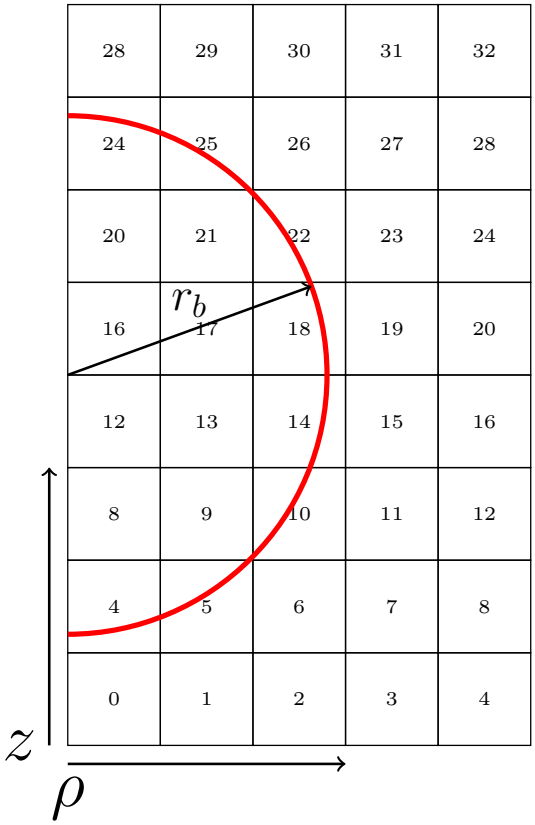


Figure 3: Scheme of a cylindrical grid decomposed over processor elements. In this case we can see that the surface only passes through some of these processors.

Our goal is to know the values of the wavefunction and its first radial derivative on the surface over a regular spherical mesh. The first derivative will be calculated by a finite difference rule. Therefore we have to interpolate the wavefunction at several radial points in order to apply the finite difference rule. In addition, initial implementations of t-SURFF in spherical coordinates used spherical harmonics for describing the angular coordinates. We do the same and so the user must specify the maximum angular momentum quantum number to use. This value allows us to determine the number of (θ, ϕ) points that will be mapped over the spherical surface. Use of the interpolation routines requires three steps in the TDSE/TDDFT solver.

S1 Initialization

The solver calls one of two routines: `initialize_cylindrical_surface` or `initialize_cartesian_surface`. It allocates and initialises the arrays to be used to interpolate the function from the working coordinates to spherical coordinates. It also creates the HDF5 surface file with it own internal folder structure. The user must provide as input parameters the mesh points for the grid, the boundary radius, r_b, the maximum angular momenta of the stored wavefunction and information on how the grid is split between processors. In order to calculate derivatives of the wavefunction using finite-differences, the finite difference rule type is specified together with the radial grid spacing Δr.

After creating a grid in spherical coordinates for all required points on the surface, the initialisation routine locates the closest surface points in the original coordinate system by finding their corresponding array indices. These array indices serves as a mapping between the original and the spherical coordinate systems.

S2 Interpolation
At each timestep during a simulation we call one of two routines: `get_cylindrical_surface` or `get_cartesian_surface`. These subroutine takes the wavefunction, time, electric field and vector potential as input parameters. The wavefunction is transformed to spherical coordinates and its first radial derivative is calculated on a spherical surface. Both quantities are then written to file.

Usually grid-based codes are parallelised by distributing the grid across cores. In this scenario, the surface will be contained on some processing cores. For example, a diagram of a parallelised grid in cylindrical coordinates is shown in figure 3. The initialisation routine determines which processors contain the surface and which do not. Only cores containing a part of the spherical surface will carry out interpolation.

S3 Finalization
At the end of the simulation we call one of two routines: `delete_surface_2D` or `delete_surface_3D`. This deallocates all allocated arrays and closes the output file.

The I/O routines create a single HDF5 file that is opened collectively by all the participating MPI processes. Each process writes its piece of the surface to this file. One of the advantages of using the HDF5 format is that each file can be split in a folder-like structure, allowing for the highly-organised collection of data. Additionally, the HDF5 format gives efficient MPI-I/O performance.

For calculation of joint electron-ion spectra the procedure is similar to above. In this case we now need to deal with two surfaces: the electronic surface (at r_s) as described above and an ionic surface (at R_s). In this case, since we only deal with vibrational motion (a 1D treatment) we do not need to perform an interpolation in the ionic coordinate. Three sets of routines are then provided to output the surface files. For THeREMIN, these routines are

- `initialize_correlated_cylindrical_surface`
- `get_correlated_cylindrical_surface`
- `idelete_correlated_surface_2D`

while for RHYthMIC we use

- `initialize_correlated_cartesian_surface`
- `get_correlated_cartesian_surface`
- `delete_correlated_surface_3D`

3.3 Implementation of the t-SURFF method

Once we have obtained the surface file from a time-dependent calculation, we can calculate the spectral amplitudes with the t-SURFF method. This can be achieved using an external program, `tsurff_calculator`, that is distributed with the library. This program processes the surface file that is output from the TDSE/TDDFT solver. When executed, this program asks the user for various parameters that were used in the calculation, creates the required arrays, reads the relevant data from the file, calculates the surface integral, integrates this in time and outputs the resulting spectral

information to disk. This output can then be visualised by another supplied python script, `popsicle_viewer`.

The parameter requested by `tsurff_calculator` at runtime are

- the name of the surface file,
- the surface radius, given in atomic units,
- the grid spacing of the k coordinate in atomic units,
- the maximum value of the k coordinate in atomic units,
- the maximum angular momentum of the calculation,
- the number of Gauss nodes in the time integral,
- if we want to transform the wavefunction from length to velocity gauge, and
- the filename for the PES file, the filename for the file file containing the spectral amplitudes integrated over angles, and the filename for the file containing the angularly resolved spectral amplitudes.

Once this information has been supplied the main calculation begins and proceeds in three stages

1. The surface information in spherical coordinates are decomposed in spherical harmonics
2. The amplitudes are then calculated according to the formula [7]

$$
\begin{aligned}
b(\boldsymbol{k})e^{-iT\boldsymbol{k}^2/2} = i\sqrt{\frac{2}{\pi}}\int_0^T dt \exp\left(\frac{i}{2}\int_0^t d\tau\,[\boldsymbol{k}+\boldsymbol{A}(\tau)]^2\right) \\
\times\sum_{lm}\Bigg[\ \frac{(-i)^l}{2}\Big(kr_b j_l'(kr_b)-j_l(kr_b)\Big)f_{lm}(r_b)Y_{lm}(\Omega_k) - \frac{(-i)^l}{2}\,r_b j_l(kr_b)f_{lm}'(r_b)Y_{lm}(\Omega_k) \\
-(-i)^{l+1}j_l(kr_b)\Bigg(\sum_{\ell\mu} A_x(t)\Big\langle Y_{lm}(\Omega_r)\Big|\sin\theta\cos\phi\Big|Y_{\ell\mu}(\Omega_r)\Big\rangle f_{\ell\mu}(r_b) \\
+A_z(t)\Big\langle Y_{lm}(\Omega_r)\Big|\cos\theta\Big|Y_{\ell\mu}(\Omega_r)\Big\rangle f_{\ell\mu}(r_b)\Bigg)\Bigg]. \qquad (18)
\end{aligned}
$$

3. For calculation the time integral we use a Gauss-Legendre quadrature.

When calculating joint electron-ion spectra using THeREMIN and RHYthMIC we use an external program called `tsurff_correlated_calculator`. In this case we read in the above parameters for the electronic surface. In addition we now read in parameters for

- the ionic surface radius in atomic units,
- the grid spacing of the k coordinate for the ionic coordinate in atomic units, and
- the maximum value of the k coordinate for the ionic coordinate in atomic units.

This code then integrates the time integrals detailed in Section III of Reference [6].

Coordinate	Number of points	Grid Spacing (Bohr)	Mesh Extent (Bohr)
ρ	500	0.1	$0 \leq \rho \leq 50$
z	500	0.1	$-25 \leq z \leq 25$

Table 1: Grid parameters using for testing the interpolation routines in 2D using THeREMIN. In this case the 2D grid is represented in cylindrical coordinates.

Interpolation method	Timing (seconds)
Shepard's method	1.1022
Bicubic interpolation	0.0296

Table 2: 2D interpolation timings for Shepard's method and bicubic interpolation from data in cylindrical coordinates onto a surface at radius r_b = 20. The given timings were taken as the average of 10 runs on ARCHER. The grid parameters used are detailed in Table 1.

3.4 Implementation of the sampling point method

The raw data for the sampling point method can be obtained in the interpolation and data output phase of a t-SURFF calculation, as described in Section 3.2. In this case the set of (θ, ϕ) points on the surface at r_b represent the sampling points as described in Section 2.3. During a calculation, the wavefunction information at these sampling points are output to the surface file.

In order to calculate the PES, using Eq. (13), we use the utility routine `sampling_calculator`, that is distributed with the library. This program processes the surface file that is output from the TDSE/TDDFT solver. When executed, this program asks the user for various parameters that were used in the calculation, reads the relevant data from the file, calculates the Fourier transforms and outputs the resulting spectral information to disk. This output can then be visualised using the python script, `popsicle_viewer`.

4 Illustrative results

In this section we will present some results for the various methods implemented in POpSiCLE. In particular we are interested in the accuracy and efficiency of the various approaches. The section is arranged as follows. We first compare the interpolation routines used for the surface flux calculations. Next we compare the three approaches for calculating PES in 2D and 3D. We will present results comparing THeREMIN and RHYthMIC. We will use these two codes as we can consider the same system (H_2^+) irradiated by the same laser pulse. This will allow us to verify that the methods in 2D and 3D have been implemented correctly. In addition we will see how the various methods perform in terms of the computational resources required in order to obtain converged results.

4.1 Efficiency of interpolation routines

As described in Section (3.2), both scattered interpolation (using Shepard's method) and bi/tri-cubic interpolation are implemented. Here, we compare the efficiency of each approach in both 2D and 3D using serial calculations. In all cases we are going to consider the 3d eigenstate of the hydrogen atom (i.e. the state with n = 3, l = 2 and m = 0) and interpolate this on a surface of given radius. This

Coordinate	Number of points	Grid Spacing (Bohr)	Mesh Extent (Bohr)
x	121	0.1	$-6 \leq x \leq 6$
y	121	0.1	$-6 \leq y \leq 6$
z	121	0.1	$-6 \leq z \leq 6$

Table 3: Grid parameters using for testing the interpolation routines in 3D using EDAMAME. In this case the 3D grid is represented in Cartesian coordinates.

Interpolation method	Timing (seconds)
Shepard's method	740.1859
Tricubic interpolation	152.5022

Table 4: 3D interpolation timings for Shepard's method and tricubic interpolation from data in Cartesian coordinates onto a surface at radius r_b = 4.5. The given timings were taken as the average of 10 runs on ARCHER. The grid parameters used are detailed in Table 3.

is similar to what we will carry out in our surface flux calculations. However, in the results presented here, the computational demand is greater than in an actual simulation, due to the lack of parallelisation.

In order to test the 2D routines we use THeREMIN. In this case the 3d state of hydrogen is represented in cylindrical coordinates. A serial calculation is carried out using the grid parameters set out in Table 1. The grid used in the calculation therefore has the structure set out in Figure 3, but using only one core. For interpolation, we will consider a surface of radius r_b = 20 Bohr and we will interpolate onto 101 equally spaced θ-points on this surface. This illustrates why the computational demand is greater in these tests: in a parallel run a given core would only have to interpolate onto a small fraction of the 101 θ-points. The resulting timings for the two interpolation methods are presented in Table 2. We can see that bicubic interpolation is 37 times faster.

In order to test the 3D routines we use EDAMAME. In this case the 3d state of hydrogen is represented in Cartesian coordinates. A serial calculation is carried out using the grid parameters set out in Table 3. For interpolation, we will consider a surface of radius r_b = 4.5 Bohr and we will interpolate onto a spherical surface consisting of 101 equally spaced θ-points and 201 equally spaced ϕ-points on this surface, i.e 20301 points in total. Again, the computational demand is greater in these tests: in a parallel run a given core would only have to interpolate onto a small fraction of these points. The resulting timings for the two interpolation methods are presented in Table 4. We can see that tricubic interpolation is 4.85 times faster.

4.2 Comparison of PES approaches in 2D

In order to benchmark the different approaches in 2D we will study the interaction of H_2^+ with a linearly polarised VUV attosecond pulse. In this case only vibrational motion of the molecular ion will be considered and we consider the laser polarization to be aligned along the molecular axis. For solving the TDSE we use THeREMIN which solves the TDSE in cylindrical coordinates. The system is shown schematically in Figure 1(a) where the field and molecular axis are aligned along the z-axis. Due to the resulting cylindrical symmetry only the ρ and z coordinates need to be considered.

We consider the interaction with a 17 cycle laser pulse having wavelength λ = 17.63 nm (the photon energy is 2.58 Hartree) and peak intensity 1.0×10^{13} W/cm^2. The duration of this pulse is $\sim$ 1.0

Method	Coordinate	Points per core	Number of cores	Grid Spacing	Mesh Extent
Fourier	ρ	300	4	0.1	$0 \leq \rho \leq 117$
	z	301	9	0.1	$-135 \leq z \leq 135$
t-SURFF	ρ	300	3	0.1	$0 \leq \rho \leq 84$
	z	301	6	0.1	$-87 \leq z \leq 87$

Table 5: Grid parameters using for calculating the PES using THeREMIN using the Fourier method and the surface flux (t-SURFF) methods in 2D. For the Fourier calculation, the code was parallelised using 36 cores while for the surface flux method 18 cores were used. Both calculations were carried out on a local cluster at QUB. This system consists of Intel(R) Xeon(R) CPU E5-2640 @ 2.50GHz, connected by an Infiniband switch. All distances are given in Bohr radii.

Method	Number of cores	Calculation time (minutes) (minutes)	Time per timestep (seconds)	Total resource (core hours)
Fourier	36	110	0.97	66
t-SURFF	18	79	1.05	24

Table 6: Timings for time-evolving the TDSE using THeREMIN for the Fourier method and the surface flux (t-SURFF) methods in 2D. The grid parameters for each simulation are those given in Table 5. As well as the total time, we present the time taken per timestep since both calculations are carried out for different numbers of timesteps.

femtosecond. In this case the laser intensity is small enough that one-photon processes dominate. Therefore, absorption of one photon will ionize the molecular ion with the electron carrying away 1.48 Hartrees of energy (a wavenumber of k_e = 1.72 a.u.). By calculating the PES, we should see a single peak at this energy. Due to the short-duration of the laser pulse, the bandwidth of frequencies present in the pulse will result in a broad peak.

Consider the calculation of the PES using the Fourier method. The grid parameters used are given in the first part of Table 5. In this case we propagate the TDSE with a time step of 0.02 a.u. for the duration of the pulse plus a further 10 cycles after the pulse. The propagation after the pulse is necessary to allow ionizing wavepackets to travel sufficient distance from the parent ion in order to be clearly distinguished from bound state density. For the calculation using t-SURFF and the sampling point methods we use the grid parameters as shown in the second part of Table 5. The measuring surface was set at r_s = 50 Bohr. In this case the calculation is smaller because we use absorbing boundaries to remove electron wavepacket that passes beyond the measuring surface: we start the absorption at 60 Bohr. For this calculation we propagate the TDSE for 20 cycles after the laser pulse in order to ensure that all ionizing wavepacket has sufficient time to pass the measuring surface.

Running all three calculations produced the results shown in Figure 4. In this case we see that the the position of the peak is almost identical in all three cases. Indeed the Fourier method is slightly displaced to higher energy: this is due to the TDSE not being propagated for long enough after the interaction. However, propagating for longer would require a larger grid, thus increasing the computational cost. For the sampling point method, we see that the position of the peak is well reproduced. However, as expected, the spectral density is not correct when using this method. The resources required for the time evolution of the TDSE for the Fourier method and the surface flux method is shown in Table 6. We can clearly see that the time taken per timestep is larger for the surface flux calculation: this is due to the extra overhead in carrying out the interpolation of the

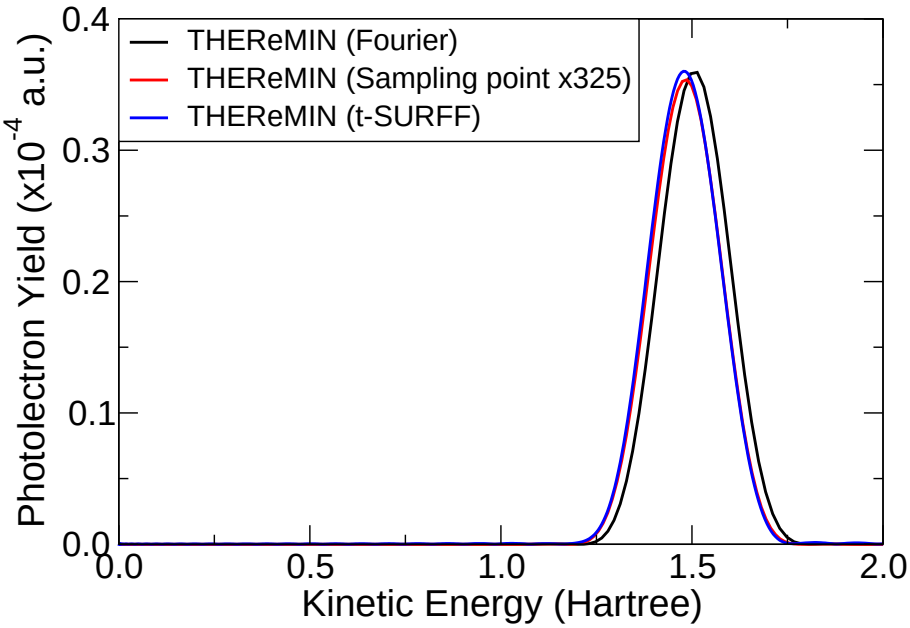


Figure 4: Photoelectron spectra for photoionization of H_2^+ after interaction with a 17 cycle laser pulse having wavelength λ = 17.63 nm and peak intensity 1.0×10^{13} W/cm^2. Results were calculated using THeREMIN using three approaches: the Fourier method, the sampling point method and the surface flux (t-SURFF) method

wavefunction onto the measuring surface. From the table we see that the total compute resource used for the Fourier method was 66 core hours, compared with 24 core hours for the surface flux method. Hence, the surface flux method requires 64% less resource than the Fourier method.

In Figure 5 we present results for the angularly resolved spectra calculated using THeREMIN. In Figure 5(a) the results are calculated using the Fourier method while in Figure 5(b) they are calculated using the surface flux method. We see that both sets of results show excellent agreement.

4.3 Comparison of PES approaches in 3D

In order to test our implementation of the different approaches in 3D we again perform the same set of calculations as carried out in Section 4.2. In this case we use RHYthMIC which solves the TDSE in 3D using Cartesian coordinates. The grid decomposition is identical to that used in EDAMAME, our TDDFT solver.

Consider the calculation of the PES using the Fourier method. The grid parameters used are given in the first part of Table 7. In this case we propagate the TDSE with a time step of 0.02 a.u. for the duration of the pulse plus a further 10 cycles after the pulse. For the calculation using t-SURFF and the sampling point methods we use the grid parameters as shown in the second part of Table 7. The measuring surface was set at r_s = 50 Bohr and absorption commenced at 60 Bohr. In this case we propagate the TDSE for 20 cycles after the laser pulse in order to ensure that all ionizing wavepacket has sufficient time to pass the measuring surface. The size of these calculations meant that ARCHER was used.

In Figure 6 we compare the Fourier and surface flux results obtained using both THeREMIN and RHYthMIC. We can see that the Fourier results agree using both codes while the surface flux results also agree. The resources required for the time evolution of the TDSE for the Fourier method and

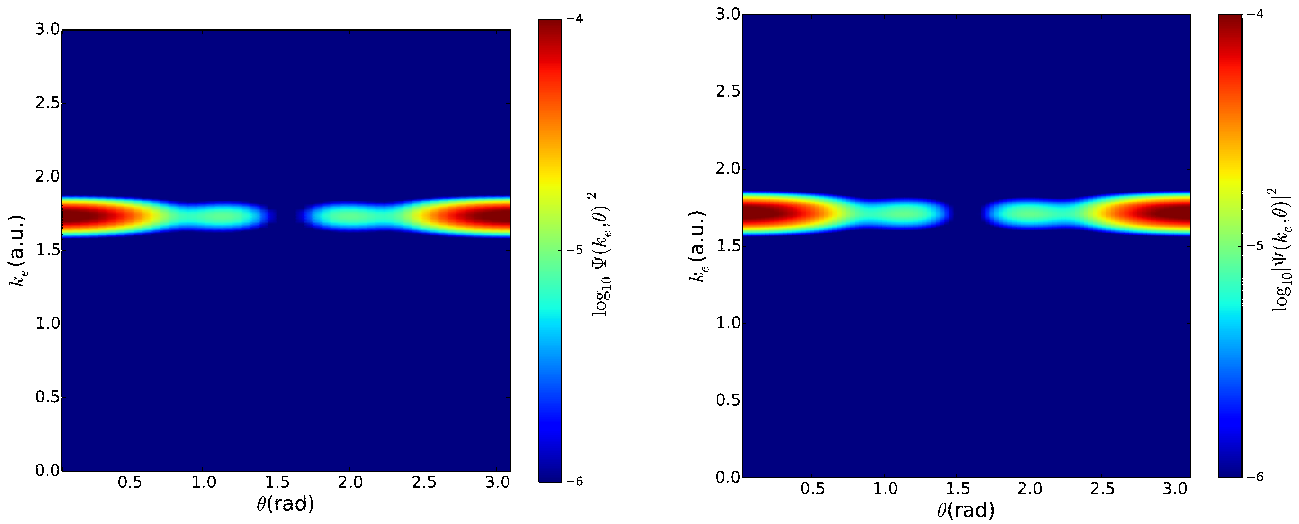


Figure 5: Angularly resolved photoelectron spectra for photoionization of H_2^+ after interaction with a 17 cycle laser pulse having wavelength λ = 17.63 nm and peak intensity 1.0×10^{13} W/cm^2. Results were calculated using THeREMIN using two approaches: (a) the Fourier method and (b) the surface flux (t-SURFF) method

Method	Coordinate	Points per core	Number of cores	Grid Spacing	Mesh Extent
Fourier	x	71	7	0.4	$-99 \leq x \leq 99$
	y	71	7	0.4	$-99 \leq y \leq 99$
	z	75	13	0.2	$-97 \leq z \leq 97$
t-SURFF	x	71	5	0.4	$-71 \leq \rho \leq 71$
	y	71	5	0.4	$-71 \leq z \leq 71$
	z	75	11	0.2	$-82 \leq z \leq 82$

Table 7: Grid parameters using for calculating the PES using THeREMIN using the Fourier method and the surface flux (t-SURFF) methods in 3D. For the Fourier calculation, the code was parallelised using 637 cores while for the surface flux method 275 cores were used. Both calculations were carried out on ARCHER. All distances are given in Bohr radii.

the surface flux method is shown in Table 8. We can clearly see that the time taken per timestep is larger for the surface flux calculation: this is due to the extra overhead in carrying out the interpolation of the wavefunction onto the measuring surface. From the table we see that the total compute resource used for the Fourier method was 1911 core hours, compared with 1228 core hours for the surface flux method. Hence, the surface flux method requires 36% less resource than the Fourier method.

5 Conclusions

We have implemented a parallel library for the computation of PES in both 2D and 3D. The library has a common core of routines that makes interfacing it with TDSE/TDDFT solvers straightforward. Three different methods have been implemented. From initial application of the library we see that the different methods give similar results. In addition, the surface flux method provides fully converged results using significantly fewer resources (36% less computational resource in 3D and

Method	Number of cores	Calculation time (minutes) (minutes)	Time per timestep (seconds)	Total resource (core hours)
Fourier	637	180	1.19	1911
t-SURFF	275	268	3.59	1228

Table 8: Timings for time-evolving the TDSE using RHYthMIC for the Fourier method and the surface flux (t-SURFF) methods in 3D. The grid parameters for each simulation are those given in Table 7. As well as the total time, we present the time taken per timestep since both calculations are carried out for different numbers of timesteps.

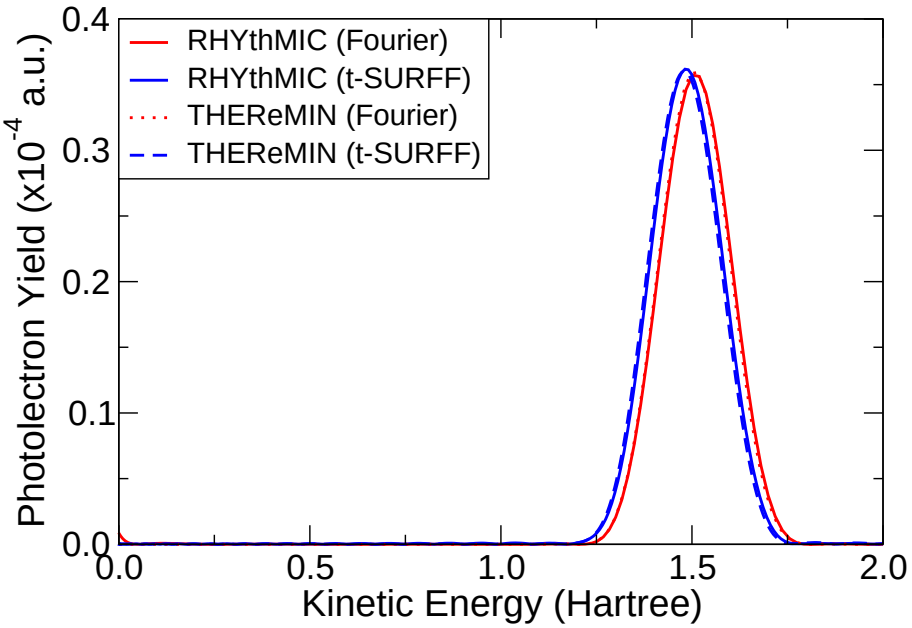


Figure 6: Photoelectron spectra for photoionization of H_2^+ after interaction with a 17 cycle laser pulse having wavelength λ = 17.63 nm and peak intensity 1.0×10^{13} W/cm^2. Results were calculated using THeREMIN and RHYthMIC using three approaches: the Fourier method and the surface flux (t-SURFF) method.

64% less resource in 2D) than using the traditional Fourier transform method.

POpSiCLE is freely available from CCPForge (https://ccpforge.cse.rl.ac.uk/gf/project/popsicle). At the minute potential users are asked to join the project in order to be able to obtain the source code.

6 Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

References

[1] G. B. Arfken and H. J. Weber. *Mathematical methods for Physicists*. Elsevier Academic Press, 6th edition, 2005.

[2] J. J. Sakurai and J. Napolitano. *Modern Quantum Mechanics*. Pearson Education, 2nd edition, 2011.

[3] L. Tao and A. Scrinzi. *New Journal of Physics*, 14:013021, 2012.

[4] A. Scrinzi. *New Journal of Physics*, 14:085008, 2012.

[5] L. Yue and L. B. Madsen. *Phys. Rev. A*, 90:063408, 2014.

[6] L. Yue and L. B. Madsen. *Phys. Rev. A*, 88:063420, 2013.

[7] A. Karamatskou, S. Pabst, Y-J Chen, and R. Santra. *Phys. Rev. A*, 89:033415, 2014.

[8] A. Pohl, P.-G. Reinhard, and E. Suraud. *Phys. Rev. Lett.*, 84:5090, 2000.

[9] MKL. Intel math kernel library. URL `https://software.intel.com/en-us/intel-mkl`.

[10] FFTW. Fastest fourier transform in the west. URL `http://www.fftw.org`.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[12] J. W. Cooley and J. W. Tukey. *Math. Comp.*, 19:297, 1965.

[13] W. I. Thacker, J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry. *ACM Trans. Math. Softw.*, 37:34:1, 2010.

[14] F. Lekien and J. Marsden. *Int. J. Num. Meth. Eng.*, 63:455, 2005.