Review of Scientific Instruments

A direct comparison of high-speed methods for the numerical Abel transform

Cite as: Rev. Sci. Instrum. **90**, 065115 (2019); https://doi.org/10.1063/1.5092635
Submitted: 13 February 2019 . Accepted: 16 May 2019 . Published Online: 25 June 2019

Daniel D. Hickstein, Stephen T. Gibson, Roman Yurchak, Dhrubajyoti D. Das, and Mikhail Ryazanov

Review of Scientific Instruments ARTICLE scitation.org/journal/rsi

A direct comparison of high-speed methods for the numerical Abel transform

Cite as: Rev. Sci. Instrum. 90, 065115 (2019); doi: 10.1063/1.5092635
Submitted: 13 February 2019 • Accepted: 16 May 2019 •
Published Online: 25 June 2019

Daniel D. Hickstein,[1,a)] Stephen T. Gibson,[2] Roman Yurchak,[3] Dhrubajyoti D. Das,[4] and Mikhail Ryazanov[5]

AFFILIATIONS

[1]Kapteyn–Murnane Laboratories, Inc., Boulder, Colorado 80301, USA
[2]Research School of Physics and Engineering, The Australian National University, Canberra ACT 2601, Australia
[3]Symerio, 91120 Palaiseau, France
[4]Department of Chemical and Environmental Engineering, Yale University, New Haven, Connecticut 06511, USA
[5]JILA, National Institute of Standards and Technology and University of Colorado, Boulder, Colorado 80309, USA

[a)]danhickstein@gmail.com. Questions and comments regarding PyAbel may be posted to http://github.com/PyAbel/PyAbel.

ABSTRACT

The Abel transform is a mathematical operation that transforms a cylindrically symmetric three-dimensional (3D) object into its two-dimensional (2D) projection. The inverse Abel transform reconstructs the 3D object from the 2D projection. Abel transforms have wide application across numerous fields of science, especially chemical physics, astronomy, and the study of laser-plasma plumes. Consequently, many numerical methods for the Abel transform have been developed, which makes it challenging to select the ideal method for a specific application. In this work, eight published transform methods have been incorporated into a single, open-source Python software package (PyAbel) to provide a direct comparison of the capabilities, advantages, and relative computational efficiency of each transform method. Most of the tested methods provide similar, high-quality results. However, the computational efficiency varies across several orders of magnitude. By optimizing the algorithms, we find that some transform methods are sufficiently fast to transform 1-megapixel images at more than 100 frames per second on a desktop personal computer. In addition, we demonstrate the transform of gigapixel images.

I. INTRODUCTION

The projection of a three-dimensional (3D) object onto a two-dimensional (2D) surface takes place in many measurement processes; a simple example is the recording of an X-ray image of a soup bowl, donut, egg, wineglass, or other cylindrically symmetric object (Fig. 1), where the axis of cylindrical symmetry is parallel to the plane of the detector. Such a projection is an example of a *forward* Abel transform and occurs in numerous experiments, including photoelectron/photoion spectroscopy,[1–7] the studies of plasma plumes,[8] flames,[9–14] and solar occulation of planetary atmospheres.[15–17] The analysis of data from these experiments requires the use of the *inverse* Abel transform to recover the 3D object from its 2D projection.

While the forward and inverse Abel transforms may be written as simple, analytical expressions, attempts to naively evaluate them numerically for experimental images does not yield reliable results.[18] Consequently, many numerical methods have been developed to provide approximate solutions to the Abel transform.[1–4,19–23] Each method was created with specific goals in mind, with some taking advantage of pre-existing knowledge about the shape of the object, some prioritizing robustness to noise, and others offering enhanced computational efficiency. Unfortunately, each algorithm is implemented with somewhat different mathematical conventions and with often conflicting requirements for the size and format of the input data. Additionally, the algorithms are written in different computer programming languages, and some only run on specific computing platforms. This situation makes it difficult to select the optimal Abel-transform method, since it can be very time-consuming to test multiple methods. Moreover, it is often unclear whether the observed differences between methods are intrinsic to the algorithm or simply a result of a particular software implementation.

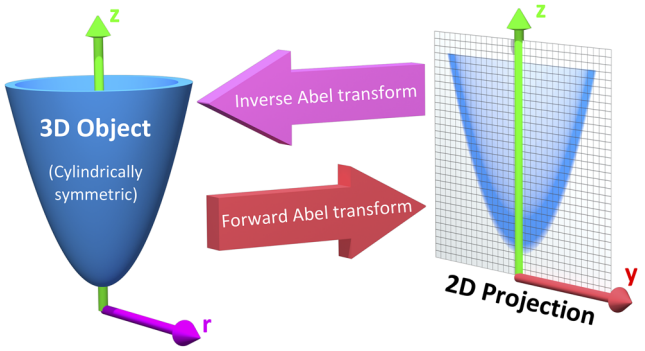

FIG. 1. The Abel transform maps a cylindrically symmetric three-dimensional (3D) object to its two-dimensional (2D) projection, a physical process that occurs in many experimental situations. For example, an X-ray image of the object on the left would produce the projection shown on the right. The *inverse* Abel transform takes the 2D projection and mathematically reconstructs the 3D object. As indicated by Eqs. (1) and (2), the 3D object is described in terms of (r, z) coordinates, while the 2D projection is recorded in (y, z) coordinates.

In this work, we present the PyAbel package, which provides a consistent interface for the Abel-transform methods via the Python programming language. Within PyAbel, the transform methods share the same mathematical conventions and data format, which allows a straightforward, quantitative comparison of the output. In addition, this package is independent of the computing platform and has been tested on Linux, MacOS, and Windows. We find that, in general, the results of the various algorithms are similar, but that some methods produce inverse Abel transforms with somewhat less noise and in better agreement with analytical solutions.

In the process of implementing these transform methods in a common language, the numerical approaches were refined to optimize efficiency. After optimization, most of the transform methods operated more quickly than the available published results would suggest, often by several orders of magnitude. Previous studies[3,21] have demonstrated "real-time" inverse Abel transforms of 1000 × 1000-pixel images at a rate of about 1 frame per second (fps), for a throughput of 1 megapixel per second (Mp/s). While this is sufficient for some applications, even inexpensive digital cameras can record and transfer high-definition video at frame rates of 30 fps or more, for a throughput of ~50 Mp/s, and specialized high-speed cameras can exceed 1000 Mp/s. Consequently, there is a need for software that can perform the inverse Abel transform at high frame rates, preferably without requiring costly supercomputing infrastructure. Such an ability could be especially powerful when the results of an Abel transform are used in a feedback loop, for example, to optimize laser pulses to control a chemical reaction.[3]

This work demonstrates the first implementation of an inverse Abel transform operating at high-definition video rates, achieved using a standard desktop computer. In addition, the inverse Abel transform of a 65 537 × 65 537-pixel image is computed, enabling Abel transforms of gigapixel images. Additionally, a side-by-side comparison of the results of numerous methods for the inverse Abel transform is presented for analytical functions and experimental photoelectron-spectroscopy data.

II. ABEL-TRANSFORM ALGORITHMS

The forward Abel transform is given by

$$F(y,z) = 2\int_y^\infty \frac{f(r,z)\,r}{\sqrt{r^2-y^2}}\,dr, \tag{1}$$

where y, r, and z are the spatial coordinates as shown in Fig. 1, $f(r, z)$ is the density of the 3D object at (r, z), and $F(y, z)$ is the intensity of the projection in the 2D plane. The inverse Abel transform is given by

$$f(r,z) = -\frac{1}{\pi}\int_r^\infty \frac{dF(y,z)}{dy}\,\frac{1}{\sqrt{y^2-r^2}}\,dy. \tag{2}$$

While the transform equations can be evaluated analytically for some mathematical functions, experiments typically generate discrete data (e.g., images collected with a digital camera), which must be evaluated numerically. Several issues arise when attempting to evaluate the Abel transform numerically. First, the simplest computational interpretation of Eq. (2) involves three loops: over z, r, and y, respectively. Such nested loops can be computationally expensive. Additionally, $y = r$ presents a singularity where the denominator goes to zero and the integrand goes to infinity. Finally, a simple approach requires a large number of sampling points in order to provide an accurate transform. Indeed, a simple numerical integration of the above equations has been shown to provide unreliable results.[18]

Various algorithms have been developed to address these issues. PyAbel incorporates eight algorithms for the inverse Abel transform, and three of these algorithms also support the forward Abel transform. Here, we focus on the results of the inverse Abel transform, because it is the inverse Abel transform that is used most frequently to interpret experimental data.

In the following, we describe the basic approach and characteristics of each transform method. The title of each method is the keyword for the method used in PyAbel. Methods that precompute matrices for a specific image size—and (optionally) save them to disk for subsequent reuse—are indicated with an asterisk (*). All methods implement the inverse Abel transform, while methods that also implement a forward transform are indicated with a superscript F (^{F}).

`basex`^{*F}—The "BAsis Set EXpansion" (BASEX) method of Dribinski and co-workers[1] uses a basis set of Gaussian-like functions. This is one of the *de facto* standard methods in photoelectron/photoion spectroscopy.[18] The number of basis functions and their width can be varied. However, following the basis set provided with the original `BASEX.exe` program, by default, we use a basis set where the full width at $1/e^2$ of the maximum is equal to 2 pixels and the basis functions are located at each pixel. Thus, the resolution of the image is roughly maintained. The `basex` algorithm allows a "Tikhonov regularization" to be applied, which suppresses intensity oscillations, producing a less noisy image.[1] In the experimental comparison presented here, the Tikhonov regularization factor is set to 200, which provides reasonable suppression of noise, while still preserving the fine features in the image.

`direct`^{F}—The "direct" methods[24] use a simple numerical integration, which closely resembles the basic Abel-transform equations (1) and (2). If the `direct` method is used in its most naive form,

the agreement with analytical solutions is poor, due to the singularity in the integral when $r = y$. However, a correction can be applied, where the function is assumed to be piecewise-linear across the pixel where this condition is met. This simple approximation allows a reasonably accurate transform to be completed. Fundamentally, the `direct` method requires that the input function be finely sampled to achieve good results. PyAbel incorporates two implementations of the direct algorithm, which produce identical results, but with different calculation speeds. The `direct_Python` implementation is written in pure Python, for easy interpretation and modification. The `direct_C` implementation is written in Cython,[25] a Python-like language that is converted to C and compiled, providing higher computational efficiency.

`hansenlaw`[F]—The recursive method of Hansen and Law[26–28] interprets the Abel transform as a linear space-variant state-variable equation, to provide a reliable, computationally efficient transform. The `hansenlaw` method also provides an efficient forward Abel transform.

`onion_bordas`—The onion-peeling method of Bordas *et al.*[2] is a Python adaptation of the MatLab implementation of Rallis *et al.*[3] While it is conceptually similar to `onion_peeling`, the numerical implementation is significantly different.

`onion_peeling`*—This method and the following two methods (`two_point` and `three_point`) are adapted from the 1992 paper by Dasch.[19] All of these methods reduce the core Abel transform to a simple matrix-algebra operation, which allows a computationally efficient transform. Dasch emphasizes that these techniques work best in cases where the difference between adjacent points is much greater than the noise in the projections (i.e., where the raw data is not oversampled). This "onion-peeling deconvolution" method is one of the simpler and faster inverse Abel-transform methods.

`three_point`*—The `three_point` method[19] provides a fast and robust transform by exploiting the observation that underlying radial distribution is primarily determined from changes in the line-of-sight projection data in the neighborhood of each radial data point. The name refers to the fact that three neighboring pixels are considered, which improves the accuracy of the method for transforming smooth functions and reduces the niose in the transformed image. The trade-off is that the ability of the method to transform very sharp features is reduced.

`two_point`*—The "two-point method" (also described by Dasch[19]) is a simplified version of the `three_point` algorithm and provides similar transform speeds. Since it only considers two adjacent points in the function, it allows sharper features to be transformed than the `three_point` method, but does not offer as much noise suppression.

`linbasex`*—The "lin-BASEX" method of Gerber *et al.*[20] models the 2D projection using spherical functions, which evolve slowly as a function of polar angle. Thus, it can offer a substantial increase in signal-to-noise ratio in many situations, but it is only appropriate for transforming projections that are appropriately described by these basis functions. This is the case for typical velocity-map-imaging photoelectron/photoion spectroscopy[4] experiments, for which the algorithm was designed. However, for example, it would not be appropriate for transforming the object shown in Fig. 1. The algorithm directly produces the coefficients of the involved spherical functions, which allows both the angular and radially integrated distributions to be produced analytically. This ability, combined with the strong noise-suppressing capability of using smooth basis functions, aids the interpretation of photoelectron/photoion distributions.

III. IMPLEMENTATION

A. Interface

PyAbel incorporates a streamlined interface to all of the transform methods, as well as numerous related functions for centering, symmetrizing, and circularizing the input images. Tools for analyzing the reconstructed images, including functions for angular and radial integration are also included. The ability to provide identical image preparation and output processing allows a quantitative comparison of transform methods.

Generating a sample image, performing a forward Abel transform, and completing an inverse Abel transform requires just a few lines of Python code:

```
import abel
im0 = abel.tools.analytical.SampleImage().image
im1 = abel.Transform(im0,
           direction='forward',
           method='hansenlaw').transform
im2 = abel.Transform(im1,
           direction='inverse',
           method='three_point').transform
```

Choosing a different method for the forward or inverse transform requires only that the `method` argument be changed. Additional arguments can be passed to the individual transform functions using the `transform_options` keyword. A basic graphical user interface (GUI) for PyAbel is also available in `examples` directory in the PyAbel repository at github.com/PyAbel/PyAbel.

In addition to the transform methods themselves, PyAbel provides many of the preprocessing methods required to obtain optimal Abel transforms. For example, an accurate Abel transform requires that the center of the image is properly identified. Several approaches allow to perform this identification in PyAbel, including the center-of-mass, convolution, and Gaussian-fitting. Additionally, PyAbel incorporates a "circularization" method, in the style of that presented by Gascooke *et al.*,[29] which allows the correction of images that contain features that are expected to be circular (such as photoelectron and photoion momentum distributions). Moreover, the `pyabel.tools` module contains a host of *post*-processing algorithms, which provide, for example, efficient projection into polar coordinates and radial or angular integration. A detailed project documentation can be found at https://pyabel.readthedocs.io.

B. Conventions

In order to provide similar results, we have ensured that the numerical conventions are consistent across the various transform methods. When dealing with pixel data, an ambiguity arises: do intensity values of the pixels represent the value of the data at $r = \{0, 1, 2, \ldots, n - 1\}$, where n is an integer, or do they correspond to $r = \{0.5, 1.5, 2.5, \ldots, n - 0.5\}$? Either convention is reasonable, but comparing results from methods that adopt differing conventions can lead to small but significant shifts. We adopt the convention that the pixel values correspond to $r = \{0, 1, 2, \ldots, n - 1\}$. One

consequence of this is that, when considering an experimental image that contains both the left and right sides of the image, the total image width must be odd, such that $r = \{1 - n, \ldots, -2, -1, 0, 1, 2, \ldots, n - 1\}$. A potential disadvantage of our "odd image" convention is that 2D detectors typically have a grid of pixels with an *even* width (for example, a 512 × 512-pixel camera). If the image were perfectly centered on the detector, our convention would not match the data, and a half-pixel shift would be required. However, in most real-world experiments, the image is not perfectly centered on the detector and a shift of several pixels is required, so the additional half-pixel shift is of no significance.

A similar ambiguity exists with regards to the left–right and top–bottom symmetry of the image. In principle, since the Abel transform assumes cylindrical symmetry, left–right symmetry should always exist, and it should only be necessary to record one side of the projection. However, many experiments record both sides of the projection. Additionally, many experiments record object that possess top–bottom symmetry. Thus, in some situations, it is best to average all of the image quadrants into a single quadrant and perform a single Abel transform on this quadrant. On the other hand, the quadrants may not be perfectly symmetric due to imperfections or noise in the experiment, and users may wish to Abel-transform each quadrant separately and select the quadrant that produces the highest-fidelity data. PyAbel offers full flexibility, providing the ability to selectively enforce top–bottom and left–right symmetry, and to specify which quadrants are averaged. By default, each quadrant is processed separately and recombined into a composite image that does not assume either top–bottom or left–right symmetry.

In our performance benchmarks, left–right symmetry is assumed because this is the most common benchmark presented in other studies.[3,21] However, the image size is listed as the width of a square image. For example, $n = 513$ corresponds to the time for the transformation of a 513 × 513-pixel image with the axis of symmetry located in the center. Since the Abel transform makes the assumption of cylindrical symmetry, both sides of the image are identical, and it is sufficient to perform the Abel transform on only one side of the image, or on an average of the two sides. So, to complete an Abel transform of a typical 513 × 513-pixel image, it is only necessary to perform the Abel transform on a 513 × 257-pixel array.

Another fundamental question about real-world Abel transforms is whether negative values are allowed in the transform result. In most situations, negative values are not physical, and some implementations set all negative values to zero. In contrast, PyAbel allows negative values, which enables its use in situations where negative values are physically reasonable. Moreover, maintaining negative values keeps the transform methods linear and gives users the option to average, smooth, or fit images either before or after the Abel transform without causing a systematic error in the baseline. Suppression of negative values can easily be achieved by including `A[A<0] = 0`.

IV. COMPARISON OF TRANSFORM METHODS

Since numerous Abel-transform methods have been incorporated into the same interface, it is straightforward to directly compare the results. Consequently, a user could simply try all of the transform methods and see which produces the best results or performance for a specific application. Nevertheless, here we present a brief comparison of the various transform methods in several cases. First, we compare the methods applied to a simple Gaussian function (for which an analytical Abel transform exists) in order to assess the accuracy of each transform method. Second, we apply each method to a synthetic function constructed of narrow peaks with noise added in order to closely examine the fundamental resolution of each method and how noise accumulates. Third, we use each method to provide the inverse Abel transform a high-resolution photoelectron-spectroscopy image in order to examine the ability of each method to handle real-world data. All calculations are completed using PyAbel version 0.8.2.[34]

The Abel transform of a Gaussian is simply a Gaussian, which allows a comparison of each numerical transform method with the analytical result in the case of a one-dimensional (1D) Gaussian (Fig. 2). As expected, each transform method exhibits a small discrepancy compared with the analytical result. However, as the number of pixels is increased, the agreement between the transform and the analytical result improves. Even with only 70 points (the case shown in Fig. 2), all the methods produce reasonable agreement. While all methods show a systematic error as r approaches zero, the `basex`, `three_point`, and `onion_peeling` seem to provide the best agreement with the analytical result. The direct methods show fairly good agreement with the analytical curve, which is a result of the "correction" discussed above. We note that the results from the `direct_Python` and the `direct_C` methods produce identical results to within a factor of 10^{-9}.

Applying the various transform methods to a synthetic function that consists of triangular peaks with one-pixel halfwidth—the sharpest features representable on the pixel grid—allows the fundamental resolution of each method to be visualized (Fig. 3). In order to provide an understanding of how each method responds to noise, the function transformed in Fig. 3 also has uniformly distributed random noise added to each pixel. The figure reveals that some methods (`basex`, `hansenlaw`, `onion_peeling`, and `two_point`) are capable of faithfully reproducing the sharpest features, while other methods (`direct`, `onion_bordas`, and `three_point`) provide some degree of smoothing. In general, the methods that provide the highest resolution also produce the highest noise, which is most obvious at low r values. The exception is the `basex` method using a moderate regularization factor [Fig. 3(b)], which exhibits low noise near the center, while still displaying good resolution. Thus, it seems that experiments that benefit from an optimal balance of noise suppression and resolution would benefit from inverse Abel-transform methods that incorporate regularization.

Applying the various inverse Abel-transform methods to an experimental photoelectron-spectroscopy image (Fig. 4) provides a comparison of how the noise in the reconstructed image depends on the transform method. To a first approximation, the results of all the transform methods look similar. The `linbasex` method produces the "smoothest" image, which is a result of the fact that it models the projection using functions fitted to the image, that vary only slowly as a function of angle. The `basex` method incorporates a user-adjustable Tikhonov regularization factor, which tends to suppress noise, especially near the symmetry axis. Here, we set the regularization factor to 200, which provides significant noise suppression while providing no noticeable broadening of the narrow features. When the regularization factor is set to zero, the `basex` method provides a transform that appears very similar to the `onion_peeling` method. For the other transform methods, the `direct` and

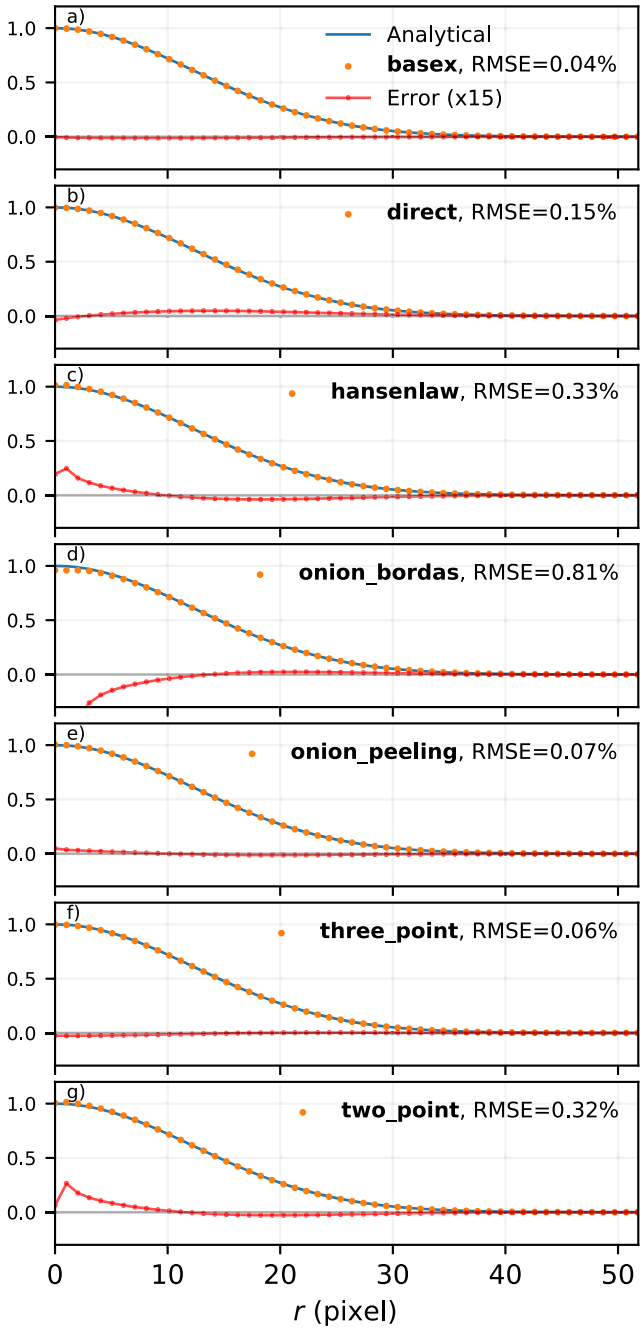

FIG. 2. Comparison of inverse Abel-transform methods for a 1D Gaussian function with 70 points. All of the inverse Abel transform methods [(a)–(g)] show reasonable agreement for the inverse Abel transform of a Gaussian function. The root-mean-square error (RMSE) for each method is listed in the figure legend. In the limit of many pixels, the error trends to zero. However, when a small number of pixels is used, systematic errors are seen near the origin. This effect is more pronounced in some methods than others. The lowest error seen from the `basex`, `three_point`, and `onion_peeling` methods. The linbasex method is not included in this figure because it is not applicable to 1D functions.

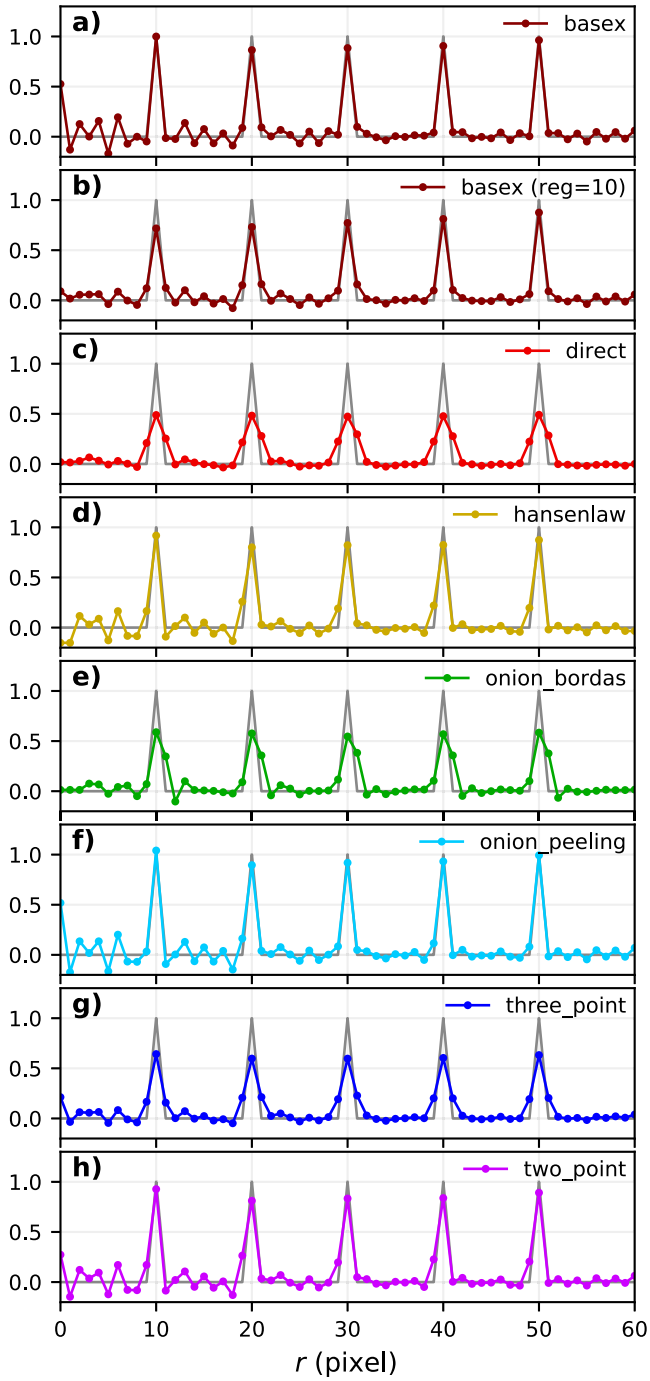

FIG. 3. Inverse Abel-transform methods applied to a synthetic image of one-pixel peaks with noise added. [(a)–(h)] The gray line represents the analytical inverse Abel transform in the absence of noise. Some methods reproduce the height of the peaks, while other methods reduce noise while somewhat smoothing the peaks. The regularization in the `basex` method provides strong noise suppression near the origin, while maintaining peak height at higher values of *r*.

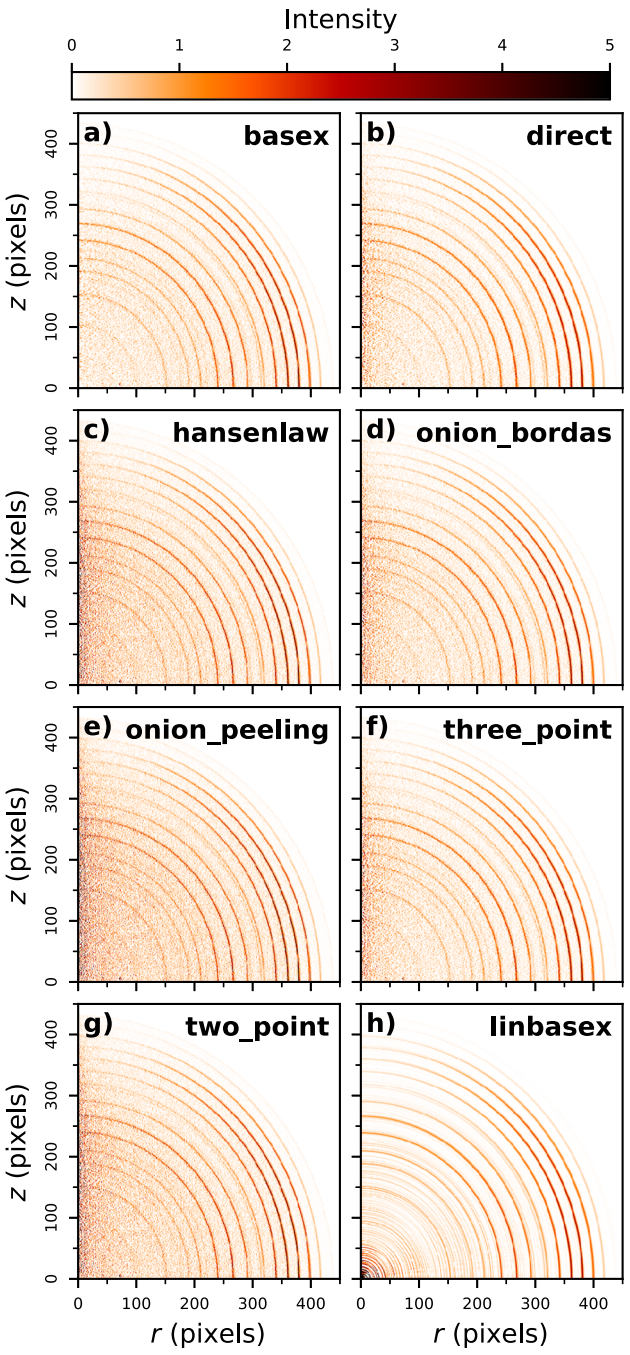

FIG. 4. Comparison of inverse Abel-transform methods for an experimental photoelectron spectrum.[30] [(a)–(h)] While all methods provide a faithful reconstruction of the experimental image, some of them cause a greater amplification of the noise present in the original image. The `linbasex` method models the image using a basis set of functions that vary slowly as a function of angle, which strongly reduces the high-frequency noise seen in the other transform methods. Besides the `basex` method with adjustable regularization, the `direct` and `three_point` methods seem particularly suited for providing a low-noise transform. This dataset is the photoelectron spectrum of O_2^- photodetachment using a 455 nm laser, as described in Ref. 30.

`three_point` methods appear to have the strongest noise-filtering properties.

Figure 5 uses the same dataset as Fig. 4, but with an angular integration performed to show the 1D photoelectron spectrum. Good agreement is seen between most of the methods, even on a one-pixel level. Small but noticeable differences can be seen in the broadness of the peaks [Fig. 5(b)]. The `hansenlaw`, `onion_peeling` and `two_point` methods show the sharpest peaks, suggesting that they provide enhanced ability to resolve sharp features. Of course, the differences between the methods are emphasized by the very high resolution of this dataset. In most cases, more pixels per peak yield a much better agreement between the transform methods. Interestingly, the `linbasex` method shows more baseline

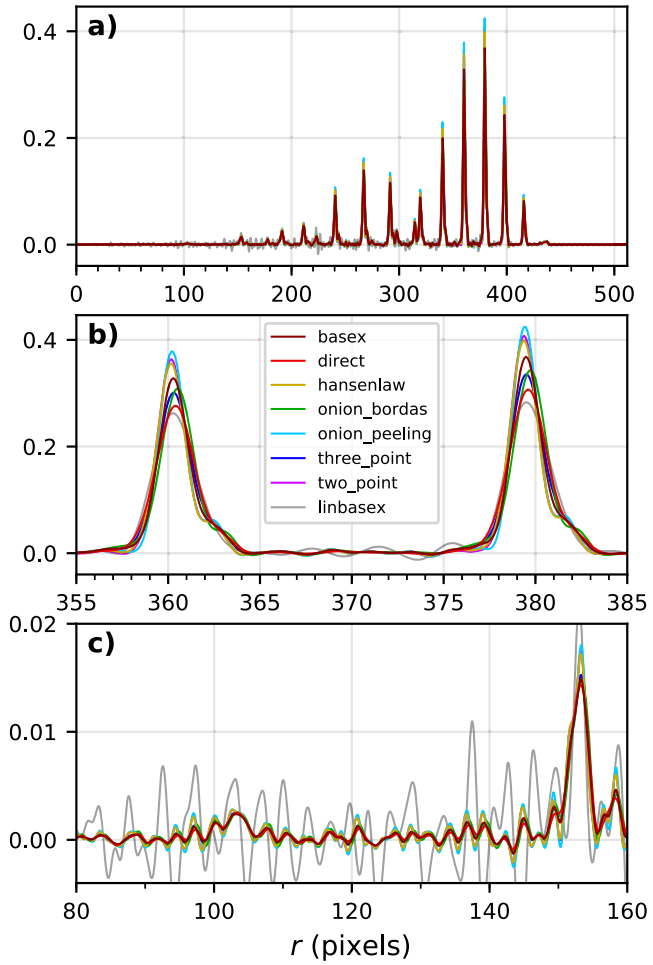

FIG. 5. Comparison of inverse Abel-transform methods applied to an experimental photoelectron spectrum and angularly integrated. The results shown in this figure are simply the angularly integrated 2D spectra shown Fig. 4. (a) Looking at the entire photoelectron speed distribution, all of the transform methods appear to produce similar results. (b) Closely examining two of the peaks shows that all of the methods produce similar results, but that some methods produce broader peaks than others. (c) Examining the small peaks in the low-energy region reveals that some methods accumulate somewhat more noise than others.

noise than the other methods. Figure 5(c) shows a close examination of the two lowest-energy peaks in the image. The methods that produce that sharpest peaks (`hansenlaw`, `onion_peeling`, and `two_point`) also exhibit somewhat more noise than the rest (except `linbasex`).

V. EFFICIENCY OPTIMIZATION

A. High-level efficiency optimization

For many applications of the inverse Abel transform, the speed at which transform can be completed is important. Even for users who are only aiming to transform a few images, the ability to perform Abel transforms efficiently may enable more effective data analysis. For example, users may want to explore many different schemes for noise removal, smoothing, centering, and circularization, and faster Abel-transform algorithms allow this parameter space to be explored more rapidly and effectively.

While PyAbel offers improvements to the raw computational efficiency of each transform method, it also provides improvements to the efficiency of the overall workflow, which are likely to provide a significant improvements for most applications. For example, since PyAbel provides a straightforward interface to switch between different transform methods, a comparison of the results from each method can easily be made and the fastest method that produces acceptable results can be selected. Additionally, PyAbel provides fast algorithms for angular and radial integration, which can be the rate-limiting step for some data-processing workflows.

In addition, when the computational efficiency of the various Abel transform methods is evaluated, a distinction must be made between those methods that can precompute, save, and reuse information for a specific image size (`basex`, `three_point`, `two_point`, `onion_peeling`, `linbasex`) and those that do not (`hansenlaw`, `direct`, `onion_bordas`). Often, the time required for the precomputation is orders of magnitude longer than the time required to complete the transform. One solution to this problem is to precompute information for a specific image size and provide this data as part of the software. Indeed, the popular BASEX application includes a "basis set" for transforming 1000 × 1000-pixel images. While this approach relieves the end user of the computational cost of generating basis sets, it often means that the ideal basis set for efficiently transforming an image of a specific size is not available. Thus, padding is necessary for smaller images, resulting in increased computational time, while higher-resolution images must be downsampled or cropped. PyAbel provides the ability to precompute information for any image size and cache it to disk for future use. Moreover, a cached basis set intended for transforming a larger image can be automatically cropped for use on a smaller image, avoiding unnecessary computations. The `basex` algorithm in PyAbel also includes the ability to extend a basis set intended for transforming a smaller image for use on a larger image. This allows the ideal basis set to be efficiently generated for an arbitrary image size.

B. Low-level computational efficiency

Transforming very large images, or a large number of images, requires inverse Abel-transform methods with high computational efficiency. PyAbel is written in Python, a high-level programming language that is easy to read, understand, and modify. A common criticism of high-level interpreted (non-compiled) languages like Python is that they provide significantly lower computational efficiency than low-level compiled languages, such as C or Fortran. However, such slowdowns can be avoided by calling functions from optimized math libraries for the key operations that serve as bottlenecks. For most of the transform methods (and indeed, all of the fastest methods), the operation that bottlenecks the transform process is a matrix-algebra operation, such as matrix multiplication. PyAbel uses matrix-algebra functions provided by the NumPy library,[31] which are, in turn, provided by the Basic Linear Algebra Subprograms (BLAS) library [for example, the open-source OpenBLAS, Intel's Math Kernel Library (MKL), or Apple's Accelerate Framework]. Thus, the algorithms in PyAbel have comparable performance to optimized C/Fortran. One subtle consequence of this reliance on the BLAS algorithms is that the performance is dependent on the exact implementation of BLAS that is installed, and users seeking the highest level of performance may wish to experiment with different implementations. In our tests, the fastest benchmarks have been achieved with MKL and Accelerate Framework.

A systematic comparison of the time required to complete an inverse Abel transform vs the width of a square image is presented in Fig. 6. We note that the `onion_bordas` method completes a transform of a 1000 × 1000-pixel image in approximately 1 s, which is nearly the same as reported by Rallis *et al.*[3] The `basex`, `two_point`, `three_point`, and `onion_peeling` methods all rely on similar matrix-algebra operations as their rate-limiting step, and consequently exhibit identical performance for typical experimental image sizes.

Figure 6(a) reveals the computational scaling of each method as the image size is increased. At image sizes below $n = 100$, most of the transform methods exhibit a fairly flat relationship between image size and transform time, suggesting that the calculation is limited by the computational overhead. For image sizes of 1000 pixels and above, all the methods show a steep increase in transform time with increasing image size. A direct interpretation of the integral for the inverse Abel transform involves three nested loops, one over z, one over r, and one over y, and we should expect n^3 scaling. Indeed, the `direct_C` and `direct_Python` methods scale as nearly n^3. Several of the fastest methods (`basex`, `onion_peeling`, `two_point`, and `three_point`) rely on matrix multiplication. These methods scale roughly as n^3, which is approximately the expected scaling for matrix-multiplication operations.[32] For typical image sizes (~500–1000 pixels width), `basex` and the methods of Dasch[19] consistently out-perform other methods, often by several orders of magnitude. Interestingly, the `hansenlaw` algorithm exhibits a nearly n^2 scaling and should outperform other algorithms for large image sizes. While the `linbasex` method does not provide the fastest transform, we note that it analytically provides the angular-integrated intensity and anisotropy parameters. Thus, if those parameters are desired outcomes—as they often are during the analysis of photoelectron spectroscopy datasets—then `linbasex` may provide an efficient analysis.

The basex, `two_point`, `three_point`, and `onion_peeling` methods run much faster if appropriately sized basis sets have been pre-calculated. For the `basex` method, the time for this precalculation is orders of magnitude longer than the transform time. For the

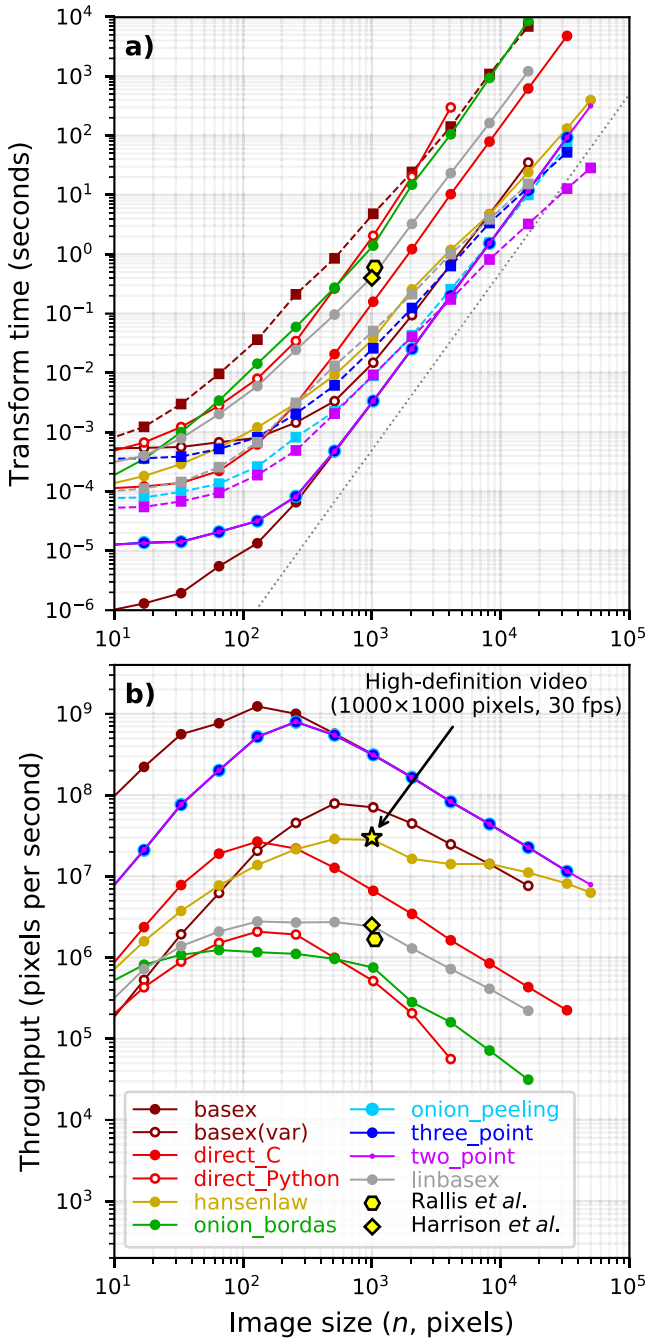

FIG. 6. Computational efficiency of inverse Abel-transform methods. (a) The time to complete an inverse Abel transform increases with the size of the image. The times for the generation of basis sets are shown with squares and dotted lines. Most of the methods display a roughly n^3 scaling (dotted gray line). (b) Alternatively, the performance can be viewed in terms of pixels-per-second rate. Here, it is clear that some methods provide sufficient throughput to transform images at high-definition video rates. These benchmarks were completed using a personal computer equipped with a 3.4 GHz Intel i7-6700 processor. For comparison, the transform times reported by Harrison *et al.*[21] (0.4 s for $n = 1000$) and Rallis *et al.*[3] (0.6 s for $n \approx 1054$) are also shown in panels (a) and (b).

Dasch methods (`three_point`, `onion_peeling`, and `two_point`), the precalculation is significantly longer than the transform time for image sizes smaller than 2000 pixels. For larger image sizes, the precalculation of the basis sets approaches the same speed as the transform itself. In particular, for the `two_point` method, the precalculation of the basis sets actually becomes faster than the image transform for $n \gtrsim 4000$. For the `linbasex` method, the precalculation of the basis sets is consistently faster than the transform itself, suggesting that the precalculation of basis sets is not necessary for this method.

Using a desktop computer (equipped with a 3.4 GHz Intel i7-6700 processor and 32 GB RAM, and running Linux) we are able to complete inverse Abel transforms with image sizes up to 50 001 × 50 001 pixels (Fig. 6), for an image size of 2.5 gigapixels using both the `hansenlaw` and `two_point` methods. For image sizes larger than this, the available RAM was filled and the transforms slowed as they moved into virtual memory. Furthermore, using a computer equipped with two 14-core 2.6 GHz Xeon E5-2697-v3 processors and 270 GB of memory, we were able to use both the `hansenlaw` and `three_point` methods to complete transforms up 65 537 × 65 537 pixels, for an image size of 4.30 gigapixels. To our knowledge, this is the first demonstration of the numerical Abel transform of gigapixel-scale images. The capability of transforming large images is becoming increasingly important, as techniques for collecting gigapixel-scale images, such as mosaic imaging and large CCD arrays become more commonplace.[33]

C. High-framerate transforms

Many experiments that rely on inverse Abel transforms utilize "high-definition" video cameras that record data at a rate of more than 20 Mp/s. For example, a "720p" camera records 1280 × 720 images at 30 fps for a data-rate of 27.6 Mp/s. As shown in Fig. 6(b), the `basex`, `onion_peelng`, `two_point`, and `three_point` methods are capable of performing an inverse Abel transform at data rates of nearly 100 Mp/s for $n \approx 1000$, thus achieving Abel transforms at high-definition video rates for the first time.

Since the `basex` method has an adjustable regularization parameter, Fig. 6 also shows a "basex(var)" curve that corresponds to changing this parameter for each data frame. This requires additional computations and thus slows down the processing. Nevertheless, the throughput remains sufficient for high-definition video rates, meaning that the regularization parameter can be adjusted in real time. This capability is very helpful even for analyzing individual images, as users can vary the regularization strength and immediately observe how it affects the results.

VI. CONCLUSION

Here, we have presented the PyAbel software package for completing forward and inverse Abel transforms, which allows the numerical transformation of three-dimensional objects into their two-dimensional projection and *vice versa*. We have implemented eight different published algorithms for the inverse Abel transform and found good agreement between the results produces using the various methods. After significant optimization of the algorithms in each method, we analyzed the computational efficiency of each method and compared the scaling of the frame rate with the image

size. We have made the first demonstration of inverse Abel transforms at high-definition video frame rates (1-megapixel images at more than 30 fps). In addition, we have transformed a 65 537 × 65 537-pixel image, realizing inverse Abel transforms of gigapixel images. Moreover, all of the Abel transform methods are implemented in Python (or Cython), a high-level scripting language that allows easy modification and incorporation into other projects. This ability to easily complete efficient Abel transforms should enable new capabilities in many fields.

PyAbel is open-source and freely available at http://github.com/PyAbel/PyAbel. The PyAbel development team encourages the incorporation of new Abel-transform methods into PyAbel.

ACKNOWLEDGMENTS

We acknowledge useful feedback from Oliver Haas, Eric Hansen, Jason Gascooke, Gilbert Shih, Eric Wells, Chris Rallis, Adi Natan, Kevin Dorney, Jennifer Ellis, and Quynh Nguyen. We thank Thomas Gerber for his assistance incorporating the linbasex method.

S.T.G.'s research was supported by the Australian Research Council Discovery Project, Grant No. DP160102585.

REFERENCES

[1] V. Dribinski, A. Ossadtchi, V. A. Mandelshtam, and H. Reisler, Rev. Sci. Instrum. **73**, 2634 (2002).
[2] C. Bordas, F. Paulig, H. Helm, and D. L. Huestis, Rev. Sci. Instrum. **67**, 2257 (1996).
[3] C. E. Rallis, T. G. Burwitz, P. R. Andrews, M. Zohrabi, R. Averin, S. De, B. Bergues, B. Jochim, A. V. Voznyuk, N. Gregerson, B. Gaire, I. Znakovskaya, J. McKenna, K. D. Carnes, M. F. Kling, I. Ben-Itzhak, and E. Wells, Rev. Sci. Instrum. **85**, 113105 (2014).
[4] D. W. Chandler and P. L. Houston, J. Chem. Phys. **87**, 1445 (1987).
[5] M. Ryazanov, "Development and implementation of methods for sliced velocity map imaging. Studies of overtone-induced dissociation and isomerization dynamics of hydroxymethyl radical (CH_2OH and CD_2OH)," Ph.D. thesis, University of Southern California, 2012.
[6] F. Renth, J. Riedel, and F. Temps, Rev. Sci. Instrum. **77**, 033103 (2006).
[7] G. A. Garcia, L. Nahon, and I. Powis, Rev. Sci. Instrum. **75**, 4989 (2004).
[8] J. Glasser, J. Chapelle, and J. C. Boettner, Appl. Opt. **17**, 3750 (1978).
[9] S. D. Iuliis, M. Barbini, S. Benecchi, F. Cignoli, and G. Zizak, Combust. Flame **115**, 253 (1998).
[10] F. Cignoli, S. D. Iuliis, V. Manta, and G. Zizak, Appl. Opt. **40**, 5370 (2001).
[11] D. R. Snelling, K. A. Thomson, G. J. Smallwood, and Ö. L. Gülder, Appl. Opt. **38**, 2478 (1999).
[12] K. J. Daun, K. A. Thomson, F. Liu, and G. J. Smallwood, Appl. Opt. **45**, 4638 (2006).
[13] C. Liu, L. Xu, Z. Cao, and H. McCann, IEEE Trans. Instrum. Meas. **63**, 3067 (2014).
[14] D. D. Das, W. J. Cannella, C. S. McEnally, C. J. Mueller, and L. D. Pfefferle, Proc. Combust. Inst. **36**, 871 (2017).
[15] G. R. Gladstone, S. A. Stern, K. Ennico, C. B. Olkin, H. A. Weaver, L. A. Young, M. E. Summers, D. F. Strobel, D. P. Hinson, J. A. Kammer, A. H. Parker, A. J. Steffl, I. R. Linscott, J. W. Parker, A. F. Cheng, D. C. Slater, M. H. Versteeg, T. K. Greathouse, K. D. Retherford, H. Throop, N. J. Cunningham, W. W. Woods, K. N. Singer, C. C. C. Tsang, E. Schindhelm, C. M. Lisse, M. L. Wong, Y. L. Yung, X. Zhu, W. Curdt, P. Lavvas, E. F. Young, G. L. Tyler, and New Horizons Science Team, Science **351**(6279), aad8866 (2016).
[16] J. D. Lumpe, L. E. Floyd, L. C. Herring, S. T. Gibson, and B. R. Lewis, J. Geophys. Res. **112**, D16308, https://doi.org/10.1029/2006jd008076 (2007).
[17] I. J. D. Craig, Astron. Astrophys. **79**, 121 (1979).
[18] B. Whitaker, *Imaging in Molecular Dynamics: Technology and Applications* (Cambridge University Press, 2003).
[19] C. J. Dasch, Appl. Opt. **31**, 1146 (1992).
[20] T. Gerber, Y. Liu, G. Knopp, P. Hemberger, A. Bodi, P. Radi, and Y. Sych, Rev. Sci. Instrum. **84**, 033101 (2013).
[21] G. R. Harrison, J. C. Vaughan, B. Hidle, and G. M. Laurent, J. Chem. Phys. **148**, 194101 (2018).
[22] E. D. Micheli, Appl. Math. Comput. **301**, 12 (2017).
[23] B. Dick, Phys. Chem. Chem. Phys. **16**, 570 (2014).
[24] R. Yurchak, "Experimental and numerical study of accretion-ejection mechanisms in laboratory astrophysics," Ph.D. thesis, Ecole Polytechnique (EDX), 2015.
[25] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, Comput. Sci. Eng. **13**, 31 (2011).
[26] E. W. Hansen and P.-L. Law, J. Opt. Soc. Am. A **2**, 510 (1985).
[27] E. Hansen, IEEE Trans. Antennas Propag. **33**, 666 (1985).
[28] J. R. Gascooke, "Energy transfer in polyatomic-rare gas collisions and van der Waals molecule dissociation," Ph.D. thesis, Flinders University, SA 5001, Australia, 2000, available at: github.com/PyAbel/abel_info/blob/master/Gascooke_Thesis.pdf.
[29] J. R. Gascooke, S. T. Gibson, and W. D. Lawrance, J. Chem. Phys. **147**, 013924 (2017).
[30] M. V. Duzor, F. Mbaiwa, J. Wei, T. Singh, R. Mabbs, A. Sanov, S. J. Cavanagh, S. T. Gibson, B. R. Lewis, and J. R. Gascooke, J. Chem. Phys. **133**, 174311 (2010).
[31] S. van der Walt, S. C. Colbert, and G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011).
[32] D. Coppersmith and S. Winograd, J. Symb. Comput. **9**, 251 (1990), computational algebraic complexity editorial.
[33] D. J. Brady, M. E. Gehm, R. A. Stack, D. L. Marks, D. S. Kittle, D. R. Golish, E. Vera, and S. D. Feller, Nature **486**, 386 (2012).
[34] S. Gibson, D. D. Hickstein, R. Yurchak, M. Ryazanov, D. D. Das, and G. Shih (2019). "PyAbel/PyAbel: v0.8.2," Zenodo. https://doi.org/10.5281/zenodo.3243413.