Solving the Schrödinger Equation Using Graphical Processing Units

Johnny Minor

Washington State University

Department of Physics and Astronomy

September 19, 2014

Introduction

During the summer of 2014 with the guidance and supervision of Jared L. Aurentz and Mark G. Kuzyk I investigated different methods for computing the second order non-linear susceptibility. I first began programming the problem using Python. I then moved onto programming in MATLAB and finally in C and CUDA. We found that using the GPU(graphical processing unit) was faster at computing the solutions to the Schrdinger equation by a factor of 100. This is a good speed up for computational time. A further benefit of computing on the GPU is that they are readily accessible to the general public in comparison to a computer cluster which may be similar in computational time for these types of problems. Also, this would allow for researchers to quickly find solutions to their problems rather than relying on computational models which can frequently take hours or days to finish.

This new way of solving the Schrödinger equation is further accentuated by a more accurate way of calculating β, the hyperpolarizability – a quality of importance to elctro optical switch. We have used the Chebyshev Spectral differentiation method on the polarization function, a spectral method that offers much greater accuracy and precision in computations. This is an important aspect of the calculation of β for polarization functions which may not be so smooth, and therefore making them more difficult to approximate for a finite difference method. This is also important becausej increases confidence in the precision of the results.

This new method for solving the Schrödinger equation seems to be a promising step towards faster, and more accurate computational methods for, but not limited to, non-linear optic materials, and the quest for a β that approaches the fundamental limit. [3]

History

A nonlinear optical response is induced by a laser beam in a medium as a function of the applied electric and magnetic fields. [1]. Generally, lasers are used as they have high enough intensity to induce the displacement of charges in proportion to a nonlinear function of the electric field. For instance second-harmonic generation scales quadratically in relation to the applied optical field lead to frequency double, where for example red light is converted to green light–a common way to make green laser pointer from less

expensive red laser diodes. Stated alternatively, nonlinear optics is the interaction of a laser with a material, and the understanding of how the phenomena that result.

Nonlinear optics was born in the early 1960's when Franken *et al.* passed a ruby laser through a quartz crystal and noticed that the emitted ultraviolet radiation was in fact twice the frequency than that which entered the crystal. Building upon this rather startling discovery Maker *et al.* and Giordmaine in 1962 improved, by many orders of magnitude, the second harmonic generation efficiency by matching the phase of the fundamental and harmonic wae using compensation of the color dispersion by birefringence in an anisotropic KDP crystal. [1] From those early years in the 1960's the field of nonlinear optics has grown immensely into a mature and rich field with a broad range of applications ranging from quantum optics, nonlinear spectroscopy, and nonlinear optical materials.

Quantum Mechanical Origins of Nonlinear Optics

Nonlinear optic phenomena can be understood macroscopically using Maxwell's Equations that describe electromagnetic theory. However, quantum mechanics offers an understanding of the origins of the nonlinear, and makes numerical predictions of the nonlinear susceptibilities. [2] The basis of quantum mechanics is the Schrödinger equation. It can take a myriad of forms depending on the conditions of the problem being solved. The specific form which is of most interest to us is the time-independent Schrödinger equation in one dimension,

$$\lambda\psi(x) = \left[\frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\psi(x) \tag{1}$$

where λ is the energy eigenvalue, $\psi(x)$ is the wave function(eigenfunction), $\hbar = h/2\mathrm{m}$ is reduced Planck's constant, m is the mass of an electron, $V(x)$ is the potential operator.

However, this equation of continuous variables must be discretized to make it solvable on a computer.Therefore, the discrete grid mesh must be defined, for the finite set of x-values which the function spans. This grid mesh is frequently defined by a right endpoint, a left endpoint, and a number of points within that domain. The equispaced grid mesh is

$$\text{interval length} = (\text{right endpoint} - \text{left endpoint})/\text{number of points} \; + \; 1. \tag{2}$$

Once these points are defined, the linear operators can be created. Furthermore, once the problem has been discretized it becomes a matrix eigenvalue problem.[9] The Schrödinger equation then takes the form

$$\lambda\,\psi(x) = H\psi(x) \tag{3}$$

where λ is the eigenvalue and ψ is the corresponding eigenvector of the defined Hamiltonian.

The scheme for computing the polarization is to compute the wave function $\psi(x)$ as a function of the electric field E using the dipole potential $V(x) = -exE$ where $-ex$ is the dipole operator. For each field E, the Hamiltonian operator has unqiue eigenvectors. The ground state eigenvector is used to determine the induced dipole moment by determinging the ground state expectation value given by:

$$P(E) = \langle \mathrm{X}\psi(E)|\psi(E)\rangle\,, \tag{4}$$

where ψ is the ground state eigenvector of the Hamiltonian and X is the position operator(specified on the domain). This yields the induced dipole moment as a function of the applied electric field. The second derivative gives β according to:

$$\beta = \frac{1}{2}\frac{\partial p^2}{\partial E^2}\Big|_E = 0 \tag{5}$$

Recent Research

Professor Mark G. Kuzyk and his many collaborators have been investigating the limits of the second order non-linear susceptibility for over a decade. He originally published a paper in 2000 which derives the fundamental limit for beta which takes the form [11]

$$\beta_{\max} = \sqrt[4]{3}\left(\frac{e\hbar}{\sqrt{m}}\right)^3 \frac{N^{3/2}}{E_{10}^{7/2}} \tag{6}$$

where N is the number of electrons, E_{10} is the energy of the first excited state, m is the mass of an electron, and $\hbar$ is the reduced Planck's constant. A scale invariant quantity can be used to compare molecules of different sizes is called the intrinsic hyperpolarizability given by,

$$\beta_{int} = \frac{\beta}{\beta_{max}}, \tag{7}$$

Where β is the computed value and β_{max} is the limit of the hyperpolarizability [3]. Since 2000 researchers around the world have been attempting to find a β_{int} greater than one, and they have all fallen short of the fundamental limit.

Research Goals

The primary tool for attempting to discover a β_{int} greater than one is through computational methods. Computational Physics is a fast paced and promising realm for the future, and algorithms for solving such a Physics problem are at the forefront of the modern-era. What Jared Aurentz and I set out to do was to create an algorithm that had parameters which could be changed relatively easily, and that would correspond to a new Hamiltonian. With this newly defined Hamiltonian it would be possible to compute β quickly. This would help Professor Kuzyk's team of researchers by being able to computationally simulate many different systems without having to experimentally create them. We had hoped, if possible, to find a system which would give characteristics that would give a β_{int} value approaching unit if not larger than one. This would give chemists and experimentalists the direction for creating the physical properties analogous to the computational model.

Method

Computing In Python

At the beginning of the summer I first began investigating the hyperpolarizability β by numerically solving the Schrödinger equation in the high-level programming language Python. As I stated previously, the Schrödinger equation must be discretized in order for it to become solvable on a computer. This means

that the Schrödinger equation becomes a synthesized matrix of the derivative operator and the potential operator over a specified domain. These operators are then added together element-wise and the scalars are multiplied in. The problem can then be solved using matrix eigenvalue methods.

In order to accomplish computing β successfully I relied heavily on the additional linear algebra functions included in the Sympy library. This allowed for much quicker and easier creation and manipulation of vectors and linear operators. In order to discretize the continuous time independent Schrödinger equation a finite-difference technique was used for discretizing the second derivative operator.

For example the discretized second derivative operator using a finite difference method for discretization takes the form[9]:

$$-\frac{\partial^2}{\partial x^2} \qquad \text{becomes} \qquad \Delta = -\begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix}$$

,

and the harmonic oscillator x^2 potential takes the form

$$V(x) \qquad \textit{becomes} \qquad \widehat{V} = \begin{pmatrix} -0.6 & & & \\ & -0.2 & & 0 \\ & & 0.2 & \\ & 0 & & 0.6 \end{pmatrix}$$

,

Where for illustration we use the interval from [-1,1] with 4 points since the endpoints are zero, they are left out. A discrete equispaced domain was also created using Python. This was needed in order to define the operators that relied upon the x values. Once the resulting Hamiltonian operator was created, then the eigenvalues and eigenvectors were computed using a familiar power iterative method for solving the matrix eigenvalue problem. The eigenvalues and eigenvectors are then used to compute β by using sum-over-states(β_{SOS})[4, 5].The sum-over-states expression requires that enough eigenvalues and eigenvectors are

used for the sum to converge usually, about 15 states are sufficient for β to converge, which can be tested by comparing numerical with analytical results [4].This is an important difference between the derivative method β_{SOS}) This of course gave the familiar value of $\beta_i nt = 0.57$ for the clipped quantum harmonic oscillator potential(x^2).

Computing In MATLAB

After I had a feel for computing β we moved to computing it using MATLAB. Thanks to the hard work of Nick Trefethen and his team of developers at the University of Oxford the use of the Chebfun package was incalculably beneficial [6].

One of the most prominent differences between Chebfun and other methods of finite differences, is that it instead uses spectral interpolation. This requires much fewer points, and as we will see, is much more accurate than a finite difference method[7].

Using Chebfun gives an intuitive and simple way of programming differential equations, such as the Schrödinger equation, in MATLAB. A quick example of the power and simplicity of Chebfun is that it's possible to compute the solutions to a simplified Schrödinger equation with only four lines of MATLAB code as follows:

```
L = 10;
dom = [-L,L];
d2 = chebop(@(x,u) -diff(u,2) + x.^2.*u, dom, 'dirichlet');
[V,D] = eigs(d2,1,0); plot(V(:,1)); norm(V(:,1),2);
```

It is important to notice that Chebfun bypasses the need to create matrices explicitly. It has this process automated. It interpolates with increasing number of Chebyshev points until the accuracy converges to machine precision(1×10^{-16}). This is incredibly handy because the interpolated matrices will not be sparse matrices like the Δ and $\hat{V}$ matrices in Python. This incredible feature can be leveraged rather nicely to compute β efficiently. However, using polynomial interpolation through Chebyshev points(hence the name 'Chebfun'), and the associated grid mesh we were able to retain twice as much fidelity in our computation in comparison with that of using Python. When using the finite difference method we would only be able

compute solutions to the Schrodinger equation with about six significant figures whereas using spectral methods we were able to compute eigenvectors to machine precision. Since we were able to compute the eigenvalues and eigenvectors so precisely we were also able to carry this precision further in the computation. This was very beneficial because data is lost when a derivative is taken which is compounded when taking the second derivative. This meant that we would be more accurate in our calculation of β. In order to match this precision using a finite difference method would not be practical because the domain would have to be increased to an astronomically small grid mesh that would correspond to a massive number of points, and a problem of such a magnitude is seemingly not computational feasible.

For comparison we created a similar algorithm to optimize β_{int} using a similar method to that which Zhou, Kuzyk and Watkins implemented in 2007.[11] Instead this time we used spectral methods instead of quadratic finite element and optimized using the same Nelder-Mead simplex optimization function call in MATLAB. This test yielded similar results to those found by Zhou, Kuzyk, and Watkins. However, the solutions are more precise by about 2 significant figures.

Computing In C

Finally, we programmed the discrete time independent Schrödinger equation using C, a low-level language that offers many hurdles for a beginner programmer. We were able to overcome these challenges. Not only was the manipulation of data a challenge but also many(nearly all) of the packages I had grown to know and love in the higher-level languages such as Python and MATLAB had to be written by myself. They had to be engineered in such a way to take an arbitrary size array. This meant learning about dynamic memory allocation. Furthermore, I had to create numerous functions such as: trapezoidal integration, inner product, normalization of an array, creating the laplacian operator, the potential operator, and the electric field operator. One by one the functions were created, tested (de-bugged), and implemented into the final program.

Where things became even more interesting was through the differentiation of the polarization function. Since we used a spectral method interpolation for approxmating the function we had to use spectral method differentiation matrix for computing β instead of using a more traditional route[7]. With the spectral method

differentation matrix code created we were able to take the second derivative of the polarization function, which would in turn give us β for the associated Hamiltonian.

Implimentation Of Spectral Methods

Spectral methods are rooted in Chebyshev interpolation. Where a function is approximated using specific points and interpolating through them with a polynomial.With spectral methods, instead of using an equispaced grid of points you must generate the points on the domain. Picking the grid points in such a manner is what truly makes spectral methods special. This method has been developed extensively from [7, 8] as well as from [10].

The Chebyshev grid points can be defined as follows on an interval from [-1,1]:

Definition 1 (Chebyshev points of the first kind)**.**

$$x_j = \cos(j\pi/n), \qquad 0 \leq j \leq n, \tag{8}$$

where n is the number of chebyshev points and j is the index value. The optimal number of Chebyshev points, which is outside of the scope of this humble report and won't be discussed, is specified to be $2^n + 1$. For instance, the number of chebyshev points you would want would be $2^1 + 1$ or $2^2 + 1$ and so on. Hence why I will use three and five Chebpoints below. This is also important considering we would need an odd number of points in order to confidently compute the derivative at zero on the domain we're interest in. An arbitrary function $f(x) = -\sin(7x)^{\cos(3x-1)} + 5x + 1$ is used for illustrative purposes to highlight just how quickly the Chebyshev interpolation converges to an arbitrary function.

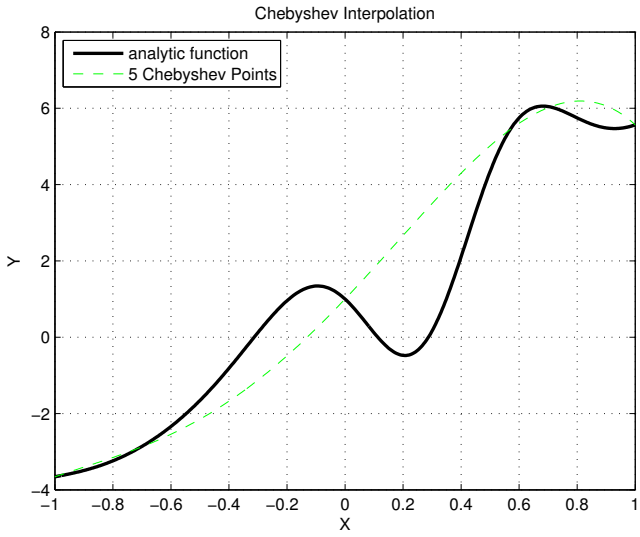

Figure 1: Interpolant using Five Chebyshev Points.

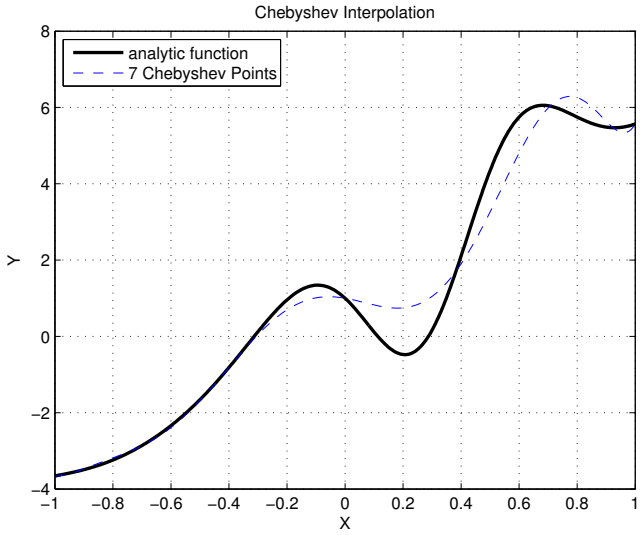

Figure 2: Interpolant using Seven Chebyshev Points.

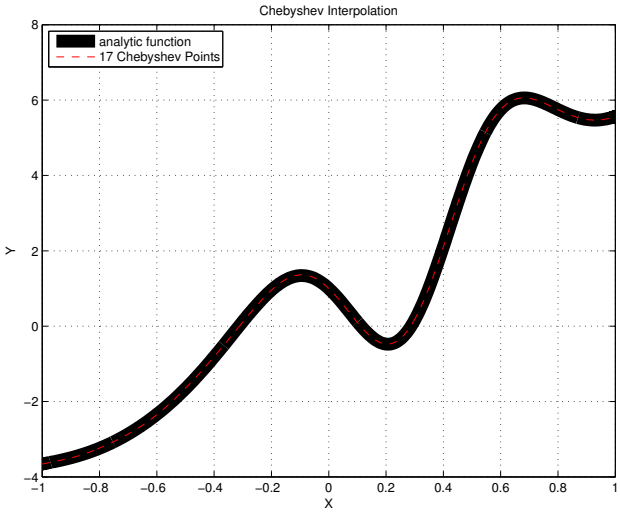

Figure 3: Interpolant using 17 Chebyshev Points.

By using a mere 17 Chebyshev points we can already hardly tell a difference visually between the analytic function and our Chebyshev polynomial interpolant. Using Chebfun we can see that after a mere 55 Chebyshev points the polynomial interpolant is at machine precision(1×10^{-16}).

With these Chebyshev points we can interpolate our computed polarization values using Lagrangian interpolation. The Lagrange form is a linear combination of Lagrange polynomials:

$$p(x) = \sum_{j=0}^{n} f_j \ell_j(x), \tag{9}$$

where x is the set of interpolation points(chebyshev points) and l_j is the jth Langrange polynomial that's unique and has the value of 1 at x_j and is 0 at all points x_k

$$\ell_j(x_k) = \begin{cases} 1, & k = j \\ 0, & k \neq j \end{cases} \tag{10}$$

This is a key element in the differentiation of computing the second derivative of the polarization function. Since we have interpolated our polarization function with Lagrangian polynomials we then can take the second derivative at x = 0 in order to find β. We use a Chebyshev differentiation matrix with entries

$$(D_N)_{00} = \tfrac{2N^2+1}{6} \qquad\qquad (D_N)_{NN} = \tfrac{2N^2+1}{6} \tag{11}$$

$$(D_N)_{jj} = \frac{-x_j}{2(1-x_j^2)} \qquad j = 1, \ldots, N-1 \tag{12}$$

$$(D_N)_{ij} = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(x_i - x_j)}, \qquad i \neq j, \quad i, j = 0, \ldots, N, \tag{13}$$

where

$$c_i = \begin{cases} 2, & i = 0 \text{ or } N \\ 1, & \text{otherwise.} \end{cases} \tag{14}$$

By constructing this matrix and multiplying it by itself element-wise we get the second derivative spectral matrix. Taking the product of it with the vector full of polarization values we are able to compute β.

Computing With A GPU

Graphical Processing Units were primarily used for video games. They were intended to manipulate the vector rendered graphics quickly and efficiently. This fact can be leveraged for scientific computing. In order to do this with NVIDIA GPU's requires a low level language named CUDA. CUDA is syntactically quite similar to modern C++ standard. However, memory management plays a key role, and programming in CUDA requires the programmer be cognizant of whether the data is on the CPU or on the GPU. Without careful attention the already tricky programming can become an arduous process of debugging. CUDA has a handful of elementary functions built-in for manipulating linear algebra type data.

One aspect of computational physics is that increased accuracy and precision requires an increase in the amount of data which is being processed. In a linear system this can often approach the millions of unknowns using a finite difference approximation scheme. However, the computational time for such a calculation can often take hours, and perhaps even days depending on the calculation or simulation being run.

This is where computations on the GPU truly shine. The GPU can take problems that seem incomputable and making implementation possible and taking only a few minutes. This is greatly advantageous because researchers can get results quickly.

Previously we had used LAPACK's eigen-solver to compute the solutions to the discretized Schrödinger equation, but with the introduction of Jared's recently completed eigen-solver we were able to implement his packages and begin off-loading much of the computations to the GPU. This reduced computational times dramatically.

We found implementing Jared's eigen-solver dramatically reduced the time. We can now compute much larger problems that we previously thought would be nearly impossible.

The results mean that we can compute the eigenstates of the quantum harmonic oscillator very fast and with relatively good accuracy. This seems very promising for larger problems with hundreds of thousands of points. the error could be brought down significantly if the spectral interpolation was written for C instead of using finite difference.

Conclusion

We found implementing Jared's eigen-solver dramatically reduced the time. We can now compute much larger problems that we previously thought would be nearly impossible.

The results mean that we can compute the eigenstates of the quantum harmonic oscillator very fast and with relatively good accuracy. This seems very promising for larger problems with hundreds of thousands of points. the error could be brought down significantly if the spectral interpolation was written for C instead of using finite difference.

Optimistically the goal was set to create an algorithm to test many different Hamiltonians and check the corresponding β. Currently we are still working on implementation on the GPU in computing β, and we are nearing a first milestone of finally implementing the C code which I wrote with Jared's eigensolver, but at this time I unfortunately don't have any concrete numerical examples. Although this goal was not reached a prodigious amount of progress was made not only regarding this project, but also for my own understanding of Professor Kuzyk and his collaborators work as and their contributions to the field of nonlinear optics.

All of these problems have been elementary, but the focus of the research this summer was to develop new ways of solving the Schrödinger equation. As we have found the new method of parallelization of solving the Schrödinger equation yields an impressive advantage of speed. Although this is only one instance it could be

a promising alternative if it could be implemented in a higher level language and therefore making it more accessible to researchers. Furthermore, the range of applications is massive considering differential equations are used from engineering to the social sciences and throughout. With the amount of data increasing rapidly in today's world it is ever-more important to find solutions for dealing with this stupendous amount of data.

Bibliography

[1] N. Bloembergen. Nonlinear optics: past, present and future. In *Guided Wave Nonlinear Optics*, pages 1–9. Springer, 1992.

[2] R. W. Boyd. *Nonlinear Optics*. Academic press, 2003.

[3] M. G. Kuzyk. Physical limits on electronic nonlinear molecular susceptibilities. *Physical review letters*, 85(6):1218, 2000.

[4] M. G. Kuzyk. Compact sum-over-states expression without dipolar terms for calculating nonlinear susceptibilities. *Physical Review A*, 72(5):053819, 2005.

[5] B. J. Orr and J. F. Ward. Perturbation theory of the non-linear optical polarization of an isolated system. *Molecular Physics*, 20(3):513–526, 1971.

[6] N. Hale T. A. Driscoll and L. N. Trefethen. *Chebfun Guide*. Pafnuty Publications.

[7] L. N Trefethen. *Spectral Methods in MATLAB*, volume 10. Siam, 2000.

[8] L. N. Trefethen. *Approximation Theory and Approximation Practice*. Siam, 2013.

[9] D. S. Watkins. *Fundamentals of Matrix Computations*, volume 64. John Wiley & Sons, 2004.

[10] B. D. Welfert. Generation of pseudospectral differentiation matrices i. *SIAM Journal on Numerical Analysis*, 34(4):1640–1657, 1997.

[11] J. Zhou, U. B. Szafruga, D. S. Watkins, and M. G. Kuzyk. Optimizing potential energy functions for maximal intrinsic hyperpolarizability. *Physical Review A*, 76(5):053831, 2007.