Numerical Approaches for Inverse Scattering

Megan Benkendorf
Advisor: Dr. Jason Murphy
Mathematics and Statistics

April 2023

Abstract

The inverse scattering problem looks at determining the potential of the time-independent Schrödinger equation given the reflection coefficient (and possibly other spectral data, if bound states are present). This is a classical problem that has been studied from a theoretical perspective. The numerical implementation of solution techniques is an interesting problem that has not been studied as closely. This project focused on the use of numerical methods to implement solutions to the inverse scattering problem. Results are shown from a neural network approach of using a python library with known scattering data and potentials.

1 Introduction

The direct and inverse scattering problems are well-studied mathematical topics widely applicable in many fields, including areas like physics, geophysics, and imaging. These problems have been studied in detail from a mathematical perspective and there are a number of theoretical approaches to resolving the inverse scattering problem. This project looks at solving both the direct and inverse problems numerically.

The direct problem is solved essentially by a finite difference approximation to the appropriate ordinary differential equation. Two approaches were taken to address the inverse problem: one, to write Matlab code to carry out an iteration scheme related to so-called Marchenko solution to the inverse scattering problem, and two, to use the Python Torch library to approximate the inverse scattering transform with a neural network. The first method did not work well, due to the fact that the theoretical solution demands fairly subtle numerics (e.g. computing jumps in the derivative of a $2d$ function across the diagonal); however, the second method performed well, at least when restricted to data chosen from a similar distribution to that of the training data. Further results are discussed in Section 2.4 and 2.5.

2 Numerical Approaches

This study looks at both the direct and inverse scattering problem for one-dimensional Schrödinger operators. The direct problem requires the numerical solution to an ordinary differential equation, which can be easily programmed, while the inverse problem is more complicated to address numerically. We followed two approaches for the inverse problem. First, we tried to directly solve an integral equation and use the solution to approximate the potential. When this did not work well, we changed our approach and used a neural network approximation to the inverse scattering map.

2.1 The Direct Problem

The direct problem is defined by finding solutions to the equation

$$\psi_{xx} + (\lambda - u)\psi = 0. \tag{1}$$

Two conditions are required for this solution to be found; first, that $u(x)$ is integrable,

$$\int_{-\infty}^{\infty} |u(x)| \mathrm{d}x < \infty \tag{2}$$

and second, that the Faddeev condition

$$\int_{-\infty}^{\infty} (1 + |x|)|u(x)| \mathrm{d}x < \infty \tag{3}$$

is satisfied.

Since the potential u tends to zero as x tends to $\pm\infty$, the solutions to the differential equation above behave like combinations of $e^{\pm ikx}$ as $x \to \pm\infty$, where $k = \sqrt{\lambda}$. Here we restrict to $\lambda > 0$, which corresponds to the 'continuous spectrum'. The case $\lambda < 0$ corresponds to 'discrete spectrum'. We work with positive potentials u, in which case we do not need to consider the discrete spectrum case.

We look for a particular solution $\hat{\psi}$ that can be written as

$$\hat{\psi} \sim \begin{cases} e^{-ikx} + be^{ikx} & \text{as}\, x \to \infty \\ ae^{-ikx} & \text{as}\, x \to -\infty \end{cases} \tag{4}$$

Physically, this corresponds to a wave of frequency k propagating to the left from $x = \infty$. The coefficient $a(k)$ describes how much of the wave is transmitted through the potential, while $b(k)$ describes how much of the wave is reflected by the potential.

In the direct problem, one is given the shape of the potential $u(x)$ and computes the scattering data, for example the reflection coefficient $b(k)$. In the inverse problem, one is given the scattering data $b(k)$ (that is, the degree to which waves of frequency k are reflected for every possible frequency k) and seeks to reconstruct the potential $u(x)$.

2.2 Coding the Inverse Problem

Our initial approach was to write Matlab functions to implement the solution to both the direct and inverse scattering problems.

We started with writing a Matlab function to produce the scattering data. We wrote code to randomly produce a large set of potential functions equal to zero outside of some fixed interval. For each potential and each frequency, we used a simple finite difference scheme to approximate the solution to the ordinary differential equation with solution given by a pure complex exponential to the left of the support of the potential. For higher frequencies, we used a smaller mesh size to get a more accurate solution. Once we had computed the solution to the right of the potential, we used a least squares approach to obtain the best guess at a representation as a linear combination of e^{-ikx} and e^{ikx}. From the coefficients in this linear combination, we computed the value of the reflection coefficient at frequency k.

For the inverse problem, we attempted to follow the presentation given in [1]. This involves first solving an implicit integral equation involving the scattering data, which is naturally approached by an iteration method. From the solution to this integral equation, we computed derivatives along the diagonal and used the values to approximate the value of the unknown potential $u(x)$. We found that our approach did not work particularly well.

We believe that it is difficult to compute the solution accurately enough to be able to detect jumps across the diagonal in the derivative. We followed this up by working with a specific example (the delta potential), a function where many quantities can be computed by hand. We were then able to compare our numerical solutions at each intermediate step to see when our method breaks down.

After seeing that our first approach did not work well, we changed approaches and considered using a neural network approximation to the inverse scattering map (the function that takes in scattering data and produces the potential). This choice was inspired by the fact that neural networks are a very robust approximation technique that require very little knowledge of the structure of the function one is trying to approximate. We discuss the neural network approach in the next section.

2.3 Applying a Neural Network

A neural network is a type of function, usually programmed to a computer to deal with the number of computations, that approximates an unknown function, mapping from $\mathbb{R}^N$ to $\mathbb{R}^M$ for real N and M. The structure is repeated composition of a linear map and a nonlinear 'activation function applied component-wise. The architecture of the neural network is defined by the choice of the activation function, the number of compositions, or layers, and internal dimensions (for example, the shape of the linear maps). The neural network itself is modified by varying this architecture and choosing parameters in the linear maps. It is necessary for the neural network approximation, then, to have a strategy for choosing values for the parameters. The neural network is trained on a dataset. A gradient descent algorithm is used to minimize the cost function for the dataset.

There are a number of libraries in Python that can be used to apply these principles. In this case, the library PyTorch was used for the neural network. Jacob Hauck, a Kummer Fellow and PhD student in the Mathematics Statistics Department, was instrumental in helping to write the Python code needed to utilize these libraries. We transferred our script for producing scattering data to Python and used it to produce many random potentials and their corresponding scattering data as needed. This was represented as x for the samples of the potential and y for the scattering data samples. We then defined a fucntion F (a neural network) to represent our approximation to the inverse scattering map (sending scattering data y to the correspond potential x).

In order to train a neural network, one must produce a large dataset to be used for training and define an appropriate loss function, which quantifies how accurate our approximation is when restricted to our labeled training dataset. Our solution to the direct problem allowed us to produce a large set of training data, as described above. We used the standard 'ℓ^2 loss function', defined by

$$L = \sum_{x \in \text{dataset}} \|y - F(x)\|^2,$$

where $\| \cdot \|$ is the standard Euclidean norm in $\mathbb{R}^M$ and (x, y) represent the labeled training data. The function L can now be seen as a function of the parameters defining the neural network; we typically write $L = L(\theta)$, where θ is a vector containing the parameters.

The parameters θ are updated in order to minimize the loss function. This is done by using a method known as gradient descent, which involves replacing the current set of parameters θ_{current} with the new set of parameters

$$\theta_{\text{new}} = \theta_{\text{current}} - \eta * \nabla_\theta L\big|_{\theta_{\text{current}}}.$$

Here $\eta > 0$ is a small parameter known as the learning rate, $\nabla_\theta L|_{\theta_{\text{current}}}$ denotes the gradient of the loss function with respect to the parameters θ evaluated at the current choice of parameters. In practice, we used some slight modifications to the standard gradient descent method (for example working with smaller batches of training data for each update), and we cycled through the entire dataset 300 times in order to complete the training of the neural network parameters.

2.4 Results

2.4.1 The Direct Problem

For the direct problem, the computation of scattering data essentially reduced to the numerical solution of an ordinary differential equation. We faced no serious issues in getting the code to work for the direct problem.

To test our code, we used the special case of a 'δ potential', $u(x) = -\delta(x)$. This is defined by requiring that

$$\int_{\mathbb{R}} f(x)\delta(x)\, dx = f(0)$$

for all continuous functions f, but in general we can think of the delta potential as being approximated by a function that is highly concentrated near $x = 0$ and has total integral equal to one. For example, in the code below, we use the function

$$\tilde{u}(x) = (\tfrac{500}{\pi})^{\frac{1}{2}} e^{-500x^2}.$$

In the case of an exact delta potential, we can compute the scattering data exactly:

$$b(k) = \frac{-1}{2ik + 1}.$$

We computed the scattering data for $\tilde{u}$ numerically and compared it to the exact result for the delta potential. The results are shown in Figure 1.

2.4.2 The Inverse Problem, First Attempt

Figure 2 shows the data produced from the original Matlab code. This is not an accurate representation of the data we wished to reconstruct. We suspect that our numerical solution was not accurate enough to faithfully reproduce jumps in the derivative, which are key for the theoretical reconstruction of the potential.

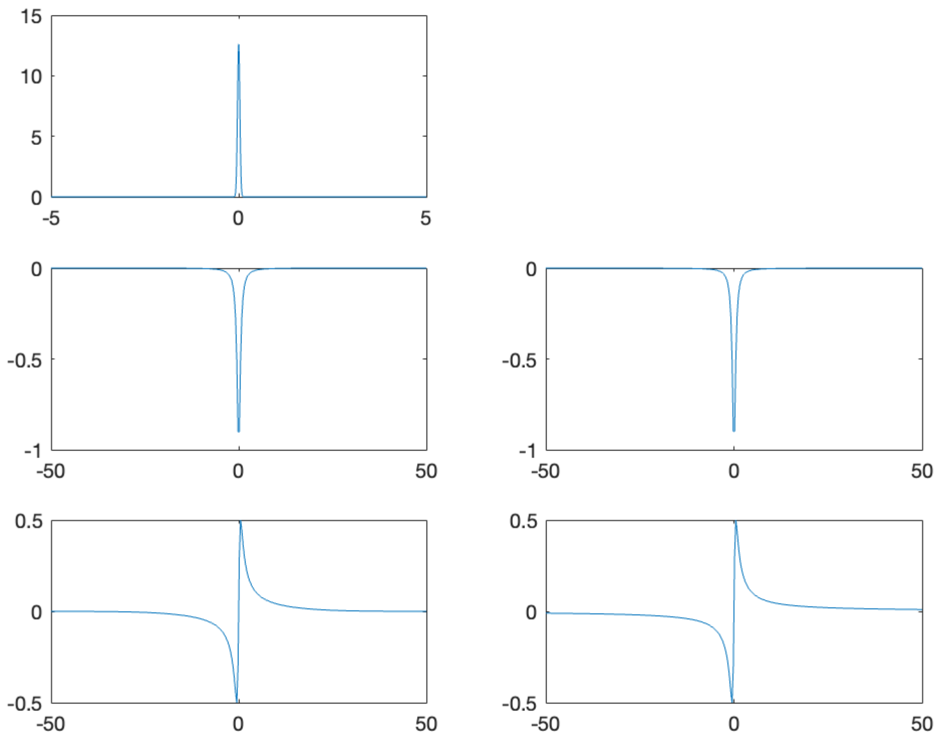
Figure 1: First row: a plot of the approximate delta potential. Second/third row: The real/imaginary part of the computed scattering data (left) versus the real/imaginary part of the true scattering data for a delta potential (right).

2.4.3 The Inverse Problem, Second Attempt

The neural network did a much better job of reproducing unknown potentials given their scattering data. To train the neural network, we selected a large number of potentials chosen randomly from a given distribution (which determined the number of bumps comprising each potential, their centers, and their decay rates). The neural network performed very well when given the scattering data corresponding to a potential chosen from this same distribution. This situation is shown in Figure 3. This figure is only a representation of many similar images the neural network was able to produce.

When given scattering data corresponding to a potential that was selected from a different distribution than the one used to train the network, the neural network did not perform nearly as well, as shown in Figure 4. For this particular set of testing potentials, the maximum bumps (previously set to five) were set to seven, and the height range of the potential (previously set to stay from zero to three) was set to stay between zero and five. This is what we expected to happen, and it shows a limitation of the neural network approach. In particular, the neural network we construct will always be strongly tied to the specific data we used to train it. However, if one has some knowledge of the types of potentials that might

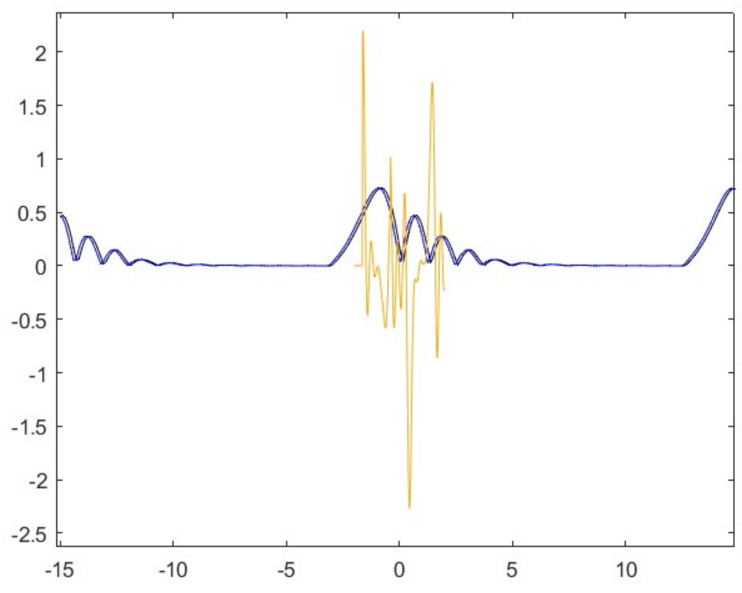

Figure 2: The Matlab code did not produce meaningful data

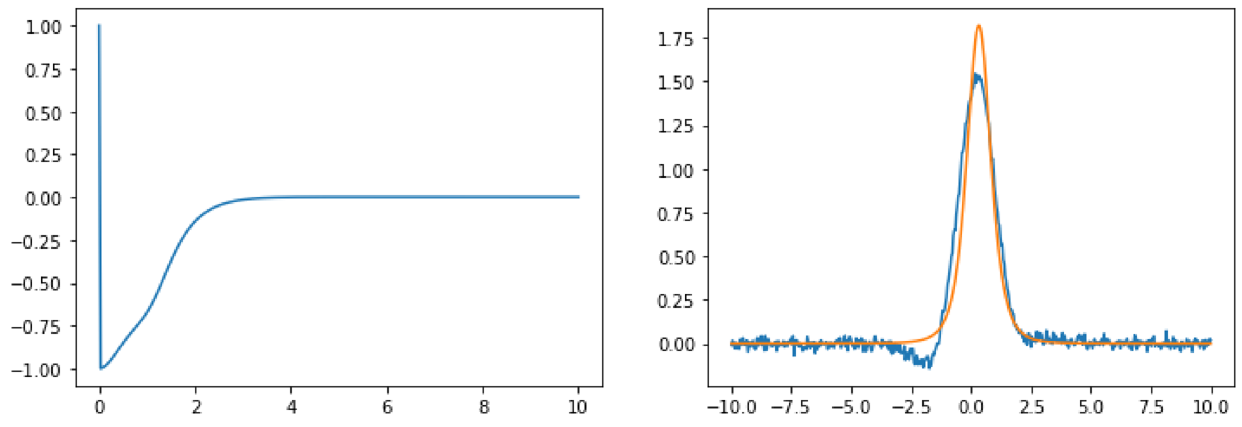

Figure 3: Potential produced by the neural network using scattering data from a potential drawn from the same distribution as that used to produce training data. Left: the scattering data, Right: the known potential is shown in blue, and the produced potential is shown in orange.

be encountered, the neural network approach could still be very useful.

2.5 Discussion

The direct problem of constructing scattering data corresponding to a given potential mostly relied on the numerical solution of an ordinary differential equation. This was fairly straightforward to implement.

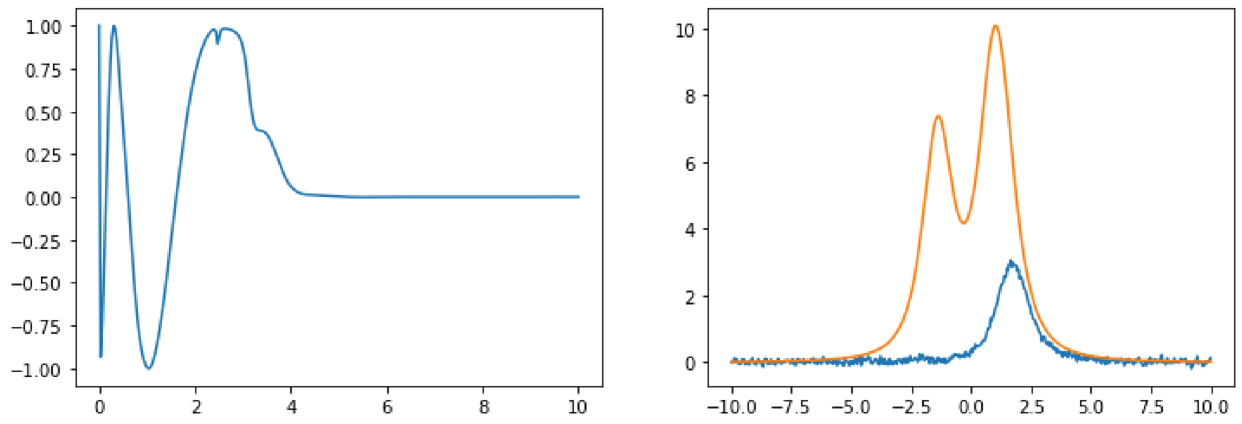

Figure 4: Potential produced by the neural network using scattering data from a potential drawn from a different distribution than that used to produce training data. Left: the scattering data, Right: the known potential is shown in blue, and the produced potential is shown in orange

In contrast, directly coding the theoretical solution to the inverse problem produced issues as the functions we construct have jump discontinuities in the derivative that cannot be easily modeled by a computer. To investigate this further, we worked with a special example (the delta potential) in which many quantities can be computed exactly. This allowed us to compare our numerical solutions with the true solution at various stages in the process, and isolate exactly where our first approach broke down.

To numerically solve the inverse problem, it was necessary to consider other methods, such as a neural network. The neural network approach is natural, as this is a robust approximation technique that does not depend on prior knowledge of the structure of the function one is hoping to approximate. The neural network was easily trainable with the Python Torch library and modeled the data well; our solution to the direct problem allowed us to produce a large set of training data. Changing parameters from those used to train the neural network does not work as well and is something that would benefit from continued study.

In this study, we focused on positive potentials, which are guaranteed to have only continuous spectrum ($\lambda > 0$). The full scattering problem must also take into account the possibility of discrete spectrum (or eigenvalues), corresponding to $\lambda < 0$. We plan to consider this case in future work.

3 Acknowledgements

I would like to express gratitude toward Dr. Murphy for working on this project with me. I would also like to acknowledge and thank Jacob Hauck for his invaluable assistance with Python and neural networks.

References

[1] Drazin, P. G., and R. S. Johnson. Solitons: An Introduction. Cambridge Univ. Press, 1996.

OURE Research

This OURE research project is my third program in Mathematics research. I have previously worked with the First Year Research Experience (FYRE) at Missouri S&T and a Research Experience for Undergraduates (REU) program funded through the National Science Foundation at Fairmont State, West Virginia. Each time I have learned something new. The FYRE project introduced me to Mathematics research. The REU allowed me time to work on research full time and see a project move quickly. This OURE experience gave me many benefits.

My project in the OURE program has helped me work towards understanding more about research in mathematics. Mathematics isn't a laboratory science. One cannot start a plant or mix chemicals. One must first find a problem they are interested in and understand the background of the project. Finding and reading literature on the subject can be very important. After the problem is understood, there are usually two aspects of mathematical research: one, to look at the problem theoretically and on paper, and two, to program a computer to model the theoretical results or solve it numerically. These two parts may be in the same project, or one may find one without the other. The former often includes derivations and a good deal of algebra. The latter involves writing code and often produces graphs to check large amounts of data at a time.

Dr. Murphy suggested the problem for the OURE project, and it has turned out to be very interesting. He also suggested a book that had a good description of the problem. This gave me a chance to become used to reading a book rather than a digital paper and discuss it. In the past, my mentors have provided papers to find online. For this project, I was able to find material at the library and use it as a reference for trying to understand the problem and write about it.

In this project, I was able to work on a more computational mathematics problem. I worked with Dr. Murphy to write and understand code in Matlab and Python. Dr. Murphy explained the background of the project, as well as suggesting a book chapter to learn more about the topic, then we worked to write the code to solve the problem numerically. In the past, I had done work with a more theoretical concept, in which I largely worked on my own manipulating equations, so in the OURE project I was able to experience a problem solved mainly through coding.

This project has a known theoretical solution, and can be applied to examples, such as the delta potential. Examples like these give a good reference as to whether or not our proposed solution is applicable. If our solution matches what we compute mathematically for the given potential, we are going in the correct direction. If it does not match, we must look at the problem again and attempt to understand why. In our case, there were discontinuities the program could not deal with. Overall, understanding the results involves comparing the findings with known findings. If there are no contradictions, the findings should be shared.

Overall, I have enjoyed working on the OURE project. Dr. Murphy was very helpful in working with me to understand the project and make progress. Jacob Hauck, a Kummer Fellow and PhD student in the Mathematics & Statistics Department, was also instrumental in helping to write and understand the code to develop the neural network. It was very interesting to look at the problem and develop new skills to solve it.