|  |  |
| --- | --- |
| Slide | Script |
| 1 | My capstone is Algorithmic Solution of Higher Order Partial Differential Equations in Julia via the Fokas Transform Method.  My supervisor is Prof. Dave Smith. |
| 2 | Let’s first go through the title. |
| 3 | Partial differential equations, or PDE, are equations that relate an unknown function to its partial derivatives. |
| 4 | The Fokas transform method is a method to solve a certain class of PDE problems algorithmically. |
| 5 | Julia is a free, open-source, high performance language for numerical computing.  Thus, my capstone is essentially about implementing the Fokas method to solve high order PDE problems in Julia. |
| 6 | In the remaining time, I will first discuss the motivation of the project. Then I will formally define the problems that can be solved by the Fokas method. Then I will give an overview of the algorithm itself, followed by two examples showing how the library I implemented can be used. Finally, I will discuss possible directions for future development of the project. |
| 7 | We care about solving PDEs because they model various physical phenomena.  E.g., the heat equation describes heat flow, the wave equation describes motion of vibrating string, and the linear Schrodinger equation describes particle motion in a system.  In reality, problems involving PDEs are usually formulated as initial-boundary-value problems, which involve   * A PDE in temporal and spatial variables defined over a domain, * A set of boundary conditions that specify the solution at the boundary of the domain, and * An initial condition that specifies the system’s initial state at time t = 0.   IBVPs arise naturally in the study of many physical phenomena, where we model a system using a PDE and measure the system’s initial state and the system’s behavior at the spatial boundaries, forming the initial and boundary conditions. |
| 8 | Some IBVPs can be solved using classical transform pair such as the Fourier transform.  The idea is that solving PDE is hard, but if we are lucky, we can use the forward transform to turn it into an ODE, solve the ODE, and use the inverse transform to turn the ODE solution into the PDE solution. |
| 9 | For more complicated IBVPs, there is no such classical transform pair.  What people usually do in this case is to resort to some ad-hoc methods that are usually specific to the given problem and cannot be generalized to related problems. |
| 10 | This is where the Fokas method comes to our rescue. I will explain what it is later, but the idea is that the Fokas method extends the idea of classical transform to an entire class of complicated IBVPs by constructing transform pairs depending on the problems’ parameters.  And what is so complicated about these IBVPs / what makes them difficult to solve is that they can have arbitrary spatial order, meaning that the differential in the spatial variable in the PDE can have arbitrarily high order. This is really impressive about the Fokas method because those of us who have been in OPDE would recall that almost all PDEs we studied are second-order, which already require a lot of work to solve. In general, high order PDEs are less well-understood than first and second-order PDEs. So |
| 11 | The Fokas method advances our understanding of high order PDEs by providing an algorithmic procedure to solve an entire class of IBVPs of arbitrary spatial order. |
| 12 | However, it is still laborious to use the Fokas method by hand. |
| 13 | Thus, my capstone aims to implement the Fokas method as a software library that supplies various computer aid in the process of solving IBVPs.  I would like to highlight that, firstly, this is the first time that the Fokas method is implemented computationally in any generality. Secondly, I chose Julia to be the language of implementation because it is open-source; there has been a movement in the mathematical community that encourages the use of open-source platforms because they allow users to check the correctness of their functionalities. Thirdly, the kind of computer aid that my library aims to supply not only include numeric representations of mathematical objects used in the algorithm, but also their symbolic formulas; I will explain why this is important later. And lastly, to the best of my knowledge, for the class of IBVPs that the Fokas method can solve, no other solver algorithm yet exists. |
| 14 | The Fokas method can solve any well-posed IBVP that can be written in this form.  2.1a is a PDE in the spatial variable x and temporal variable t, defined over the domain (0,1) x (0,T).  2.1b is the initial condition given by a function f(x); that is, at t = 0, the solution q(x,0) as a function of x is given by f(x).  2.1c is a set of homogeneous boundary conditions that the solution needs to satisfy; that is, at any fixed time t, the solution q(x,t) as a function of x satisfies the homogeneous boundary conditions given by \Phi. |
| 15 | Here are two examples of the IBVPs solvable by the Fokas method. They are linearized KdV equations that characterize shallow water waves. They both can be written in the form just described (with n = 3, a = -i). |
| 16 | Now, onto the algorithm itself.  Suppose we are given an IBVP. We want to obtain its solution; that is, we want to go from left to right in the diagram, which is difficult.  The key to the algorithm is the construction of the Fokas transform pair F\_\lambda, f\_x. The forward transform F\_\lambda turns the PDE in x and t into an ODE in t by turning differentiation in the spatial variable x into multiplication. The ODE also comes with an appropriate initial condition that comes from the original initial condition; that is, we have an ordinary IVP, whch can be solved explicitly, and when we apply the inverse transform f\_x, we would get the solution of the original IBVP.  Now, to construct the transform pair, I would need what is called the “spatial adjoint” of the IBVP. That is, we ignore the initial condition for now, what I need is the adjoint problem of the BVP in the spatial variable x. The adjoint problem comes with an adjoint spatial differential operator and a set of boundary conditions that are adjoint to \Phi. The adjoint differential operator is given explicitly, in fact, in this case, it is the spatial differential operator itself. Finding the adjoint boundary conditions requires much more work. |
| 17 | Therefore, the algorithm is essentially broken into two parts; first, we construct the adjoint boundary conditions; then, we use the adjoint and other information about the IBVP to construct the Fokas transform pair. Then we can obtain the solution directly.  (To recap (back to slide 15), given the IBVP, we first find the adjoint boundary conditions and use it to construct the transform pair F\_\lambda, f\_x. Then we apply the forward transform F\_\lambda to the PDE and the initial condition to obtain an ODE in t and a corresponding initial condition, which form an ordinary IVP. Solving the IVP and applying the inverse transform, we get the IBVP solution.) |
| 18 | Proving that the adjoint boundary conditions exist is easy. But actually constructing the adjoint requires a lot of work.  Theorems and results that specify how to construct the adjoint boundary conditions are scattered all over the place in chapter 11 of Theory of Ordinary Differential Equations by Coddington and Levinson. This textbook essentially provides a constructive existence theorem on the adjoint, and a theorem to check whether a candidate adjoint is indeed valid. But the textbook was not written to illustrate how to construct adjoint boundary conditions. So a lot of work on my part has gone into understanding and expanding the theorems and their proofs in order to adapt these results to developing an algorithm to find adjoint boundary conditions. |
| 19 | The Fokas transform pair is an integral transform-inverse transform pair. So to construct them, we need to construct the integrand and the contours over which the integrand will be integrated.  I will not go into the technicalities here, but I would just like to highlight that in constructing the integrand, the information that goes in are the adjoint boundary conditions we just constructed, and information about the IBVP, e.g., the spatial order n. |
| 20 | After defining a bunch of objects, the integrand is given by these equations. |
| 21 | The contours are defined by these horrifying-looking formulas. |
| 22 | But they can be visualized by nice-looking pictures like this. This is a sample contour drawn by hand.  To give a general idea of what these contours are, the red and green lines are the boundaries of some sectors in the sectors, and the blue and black circles are contours around the poles of the integrand, that is, dangerous places where we may have division by zero. Thus, to construct the contours, I would start with the sector boundaries as the contours’ backbone and then deform the contours to avoid the poles of the integrand. |
| 23 | Here are some sample contours drawn by the library I implemented. The circles and bulges are the poles I simulated; we see that the contours are deformed to avoid the poles. The sample contours are for different spatial orders, n = 2, 3 |
| 24 | n = 4, 5. |
| 25 | With the integrand and the contours, we can now define the Fokas transform pair. F\_\lambda is the forward transform, f\_x is the inverse transform. Note that f\_x is an integral of the integrands over the contours we just constructed. |
| 26 | With the Fokas transform pair, the solution of the IBVP is given explicitly by these equations. |
| 27 | Now I’ll illustrate how the library I implemented can be useful by two cases of sample usage.  The first case is solving IBVPs. Suppose we want to solve this IBVP.  The first step is to write it in the standard form. In this case we have n = 2, a = 1, and the boundary conditions \Phi. |
| 28 | Then we construct the adjoint boundary conditions, which can be characterized by these two matrices.  Using the matrices, we find the integrand as follows. |
| 29 | We also compute the contours to be this.  The IBVP solution found by the library can be evaluated within 20s at any given point (x\_0, t\_0). |
| 30 | Another aspect in which my library can be helpful concerns its symbolic features.  The background is that applying the Fokas method requires two technical lemmas concerning the position of the poles of the integrands and integrands asymptotic behavior.  Thus, it is necessary to be able to obtain symbolic formulas for the integrand; otherwise we cannot prove anything if we only have numeric results.  And in the case where the integrands are complicated, we would want to get a computer to derive their symbolic formulas.  My library is able to do that, and we verified that my library gives the correct symbolic formulas using the following two problems. |
| 31 | Problem 1 is an IBVP with the KdV equation.  The integrands are given by these formulas, which are derived by hand and published in my supervisor’s paper. I verified that the symbolic formulas produced by my library are equivalent to these expressions. |
| 32 | Same for Problem 2.  These IBVPs only have spatial order 3. These formulas can get very complicated easily for higher spatial orders. |
| 33 | My capstone project is a small part of a large project that my supervisor is working on. Looking ahead, the next step of my project would be to speed up the evaluation.  In my implementation, I have already taken measures to bring down the evaluation time to within 20s, e.g., by replacing the inner integral in a double integral with its explicit formula so that it is not as expensive to compute as an iterated integral.  But to analyze the IBVP solution in-depth, we would actually want to be able to graph it. That means we need to be able to evaluate the solution at any given (x\_0, t\_0) in the order of milliseconds.  In my current implementation, I’m using Julia’s native integrator, and we concluded that evaluating the solution in less than 20s is as far as I can go with the native integrator. So to improve the evaluation speed, the idea is to build a custom integrator that is tailored to evaluating the contour integrals in the Fokas transform pair. If we know the mathematical properties of the integrand and the contour, maybe we can find out which parts of the integrand contribute significantly to the integral on which parts of the contour, and discard parts of the integrand that do not contribute significantly to the integral. In this way, we may hope to save time by eliminating trivial computations. |
| 34 | To sum up:  The Fokas method is very useful because it allows solving an entire class of IBVPs of arbitrary spatial order.  My capstone developed a library in Julia to provide computer aid in the process of applying the Fokas method to solve IBVPs. Functionalities of the library include symbolic formulas of important mathematical objects, their numeric representations, and contour visualization. The library’s code, unit tests, and documentations are available on a public GitLab repository.  Looking ahead, in order to analyze the solution graphically, it is of interest to develop a custom integrator to bring the evaluation time down to the order of milliseconds. |