UNIVERSITY OF CALIFORNIA
Santa Barbara

A Minimum Sobolev Norm Discretization Scheme for Elliptic Partial Differential Equations

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical and Computer Engineering

by

Joseph Moffitt

Committee in Charge:

Professor S. Chandrasekaran, Chair

Professor C. J. Garcia-Cervera

Professor B. S. Manjunath

September 2011

The Thesis of
Joseph Moffitt is approved:

Professor C. J. Garcia-Cervera

Professor B. S. Manjunath

Professor S. Chandrasekaran, Committee Chairperson

September 2011

A Minimum Sobolev Norm Discretization Scheme for Elliptic Partial Differential Equations

by

Joseph Moffitt

To my family.

Acknowledgements

In a sense, this thesis is really the work of everyone I have ever been in contact with, and I suppose also those whom I have never met.

I would like to give gratitude to a few people at Rutgers University who really planted in me the whole idea of going to graduate school in the first place. Ken Vaz and Vivian Ho, thank you for the encouragement and knowledge. Prof Daut, I realize now what a great professor you were. Your generosity to us as undergraduates was incredible. Jon Zrake, you also played an essential part in me getting to graduate school, I thank you for your friendship and inspiration.

Special thanks to Stefan Boeriu for the super computing assistance and to my committee members B.S. Manjunath and Carlos Garcia-Cervera for their interest and input.

Prof Shiv, I would like to especially thank you for your encouragement and commitment to this work. I have learned immensely from simply just watching you think. I am deeply grateful for that experience alone, not to mention the enormous amount of mathematical and technical knowledge you have passed on to me. In addition, I feel glad to say that beyond all the academics, I consider you foremost a friend.

Karthik, Matt, and Naveen, it has been great getting to know you as lab mates and friends. I especially thank Karthik for his willingness to help me in any technical matter.

Finally, last but not least I would like to thank my family, especially my dad. You were the one who showed me how to think. I would not be where I am without your encouragement, patience, love, and enthusiasm in sharing your knowledge.

Abstract

A Minimum Sobolev Norm Discretization Scheme for Elliptic Partial Differential Equations

Joseph Moffitt

This thesis explores the idea of setting up the numerical solution to PDEs as constrained minimization problems. The minimizations are over the local coefficients which describe the PDE in specified regions, while the constraints contain the PDE itself, including the unknown solution. The minimizations are asking that Sobolev norms of the coefficients be smallest, thus guaranteeing a certain degree of smoothness to the solution of the PDE which they describe. An equivalent minimization can be obtained which is only over the unknowns and is unconstrained. We solve it using a simple least squares minimization. This leads to the solution to the PDE which is of minimum Sobolev norm. At this point, the method shows impressive results experimentally in solving difficult Elliptic PDEs including planar div-curl boundary value problems. This thesis is a documentation of the work which has gone into exploring this method thus far.

Professor S. Chandrasekaran
Dissertation Committee Chair

Contents

Acknowledgements **v**

Abstract **vii**

List of Figures **x**

List of Tables **xi**

1 Introduction **1**
1.1 Full MSN - Advantages 1
1.2 Objectives and Outline 2
1.3 Differences with Finite Difference 3

2 Interpolation **4**
2.1 Using the monomial basis 4
2.2 Runge Phenomenon 6

3 Minimum Sobolev Norm Interpolation **14**
3.1 Intuition 14
3.2 Size of Fourier Series Coefficients 15
3.3 Picking a Sobolev Norm 17
3.4 The MSN solution 22
3.5 MSN in higher dimensions 25

4 Full MSN (FMSN) **27**
4.1 Introduction 27
4.2 FMSN Solution of PDEs 28
4.3 Sparse Matrix Indexing 35

4.4 Window Placement 37
4.5 Implementation Overview - numerical considerations 43
4.6 Numerical Results 45

5 Extensions and Future Work **50**

Bibliography **51**

Appendices **52**
.1 Global Approach - 1d Boundary Value Problem 53
.2 Numerical Considerations 55
.3 Local Approach - 1d Boundary Value Problem 56
.4 MATLAB code 58

List of Figures

2.1 Runge phenomenon, 11 samples 7
2.2 Runge phenomenon, 19 samples 8
2.3 Runge phenomenon interpolant, 5 samples, $\alpha = 1$ 10
2.4 Runge phenomenon interpolant, 5 samples, $\alpha = 100$ 11

4.1 Two adjacient windows of size L=5 and overlap O=3 39
4.2 Two adjacent windows of size L=5 and overlap O=2 40
4.3 Adjacent windows of size L=5 and overlap O=1.5 41
4.4 A set of windows of size L=6.5 and overlap O=1.5 43

List of Tables

4.1 Relative Error for MSNFD vs FMSN, 'good Helmholtz' 46
4.2 Relative Error for MSNFD vs FMSN, 'hard Helmholtz' 47
4.3 FMSN div-curl results, smooth solution 49

Chapter 1

Introduction

1.1 Full MSN - Advantages

The research group I have been involved with for the last few years, under the direction of Prof. Chandrasekaran, has been exploring Minimum Sobolev Norm (MSN) methods. Initially, these ideas were applied interpolation and approximation with the idea that Runge phenomenon could be controlled. Local and global approaches were explored, and extensive experimentation showed that the method could hold its ground in a variety of settings. With that solid foundation, various ideas for solving PDEs began to be explored.

This thesis one of the most recent ideas on how to setup a PDE solver using the MSN idea. Previously, we have explored using MSN to setup finite difference type weights on arbitrary grids. This FD method showed that a high order of convergence could be obtained on a variety of PDEs, but no proof was in sight,

as is typical of finite difference approaches. In addition, the finite difference MSN solver requires a square system containing the discretization weights, which makes some types of PDEs particularly tricky to setup.

The Full Minimum Sobolev Norm (FMSN) method discussed in this thesis shows promise in that initial numerical results reveal the order of convergence to be on par and sometimes higher than FDMSN. In addition, we are forming a least squares system, thus there is no constraint on the number of equations we must use. This allows for ease of implementation of PDEs where the well-posedness is in not simple to see. Third, the method is flexible in the way memory and computation time can be managed. All the experiments done in this thesis did not require the use of a supercomputer. Finally, a proof of convergence seems plausible, where as working directly through FDMSN, no easy paths are seen.

1.2 Objectives and Outline

Therefore, the main goal this thesis is to simply document the method, since it has not yet appeared in any of our publications. It is written in a style in that anyone with a reasonable background in linear algebra should be able to read the discussion and be able to implement it. Secondly, we aim show some initial results. The results we have thus far show a high order method, which can solve

difficult Elliptic PDEs. Finally, we provide a working code, (with comments!), so that anyone can immediately get started on solving PDEs. It works!

As a quick outline of what appears in the thesis. We first cover some basic ideas behind interpolation and what can go wrong; namely Runge phenomenon. We give a highly intuitive explanation of the Runge phenomenon, for those who are new to the subject. Those familiar with these ideas can skip directly to the section on MSN. The section on MSN is also highly intuitive, and does not require any more knowledge than the basic undergraduate engineering courses. We then see how the MSN idea can be applied to the PDE solver in this thesis which we call Full MSN. Finally, we give some results and a working MATLAB code.

1.3 Differences with Finite Difference

We call the method Full MSN, because the solution to the PDE which we are solving is setup as a constrained Sobolev Norm minimization problem of which me must minimize the full Sobolev norm. The constraints are the PDE itself. This is opposed to the MSN finite difference method, which solves for the local weights using the minimum Sobolev norm idea. We can interpret this as the weights in the FDMSN method being Runge phenomenon free, as opposed to the full solution to the PDE being Runge free as in the FMSN method.

Chapter 2

Interpolation

2.1 Using the monomial basis

Interpolation is the process of obtaining new data points from a set of given old data points. More specifically, in this discussion we mean a process of finding a function which passes through a finite set of given data values. The new data points which hope to obtain are then the values which result from evaluation of this interpolating function. In order to do this, the underlying function we are finding must be expressed in some basis. Common basis functions include polynomials, sinusoids, and rational functions. Thus the problem of interpolation is reduced to a problem where we want to find the coefficients of the polynomial that passes through all the given data points, if we so choose our interpolating function to be in the monomial basis [4].

To see exactly what is meant, we begin with the simplest possible case of interpolation in the monomial basis. Our underlying function will then be some polynomial. The solution for the coefficients can be setup as a linear system of equations. How do we pick the order of this polynomial? Intuition from linear algebra says we would like an invertible square system. This leads to choosing the order of the polynomial to be one less than the number of data points given. A formal proof of the uniqueness of the interpolating polynomial in this case can be found in any textbook on the subject. However, things will not go as planned, as we will see shortly. Solving the following system $V\mathbf{a} = \mathbf{f}$ gives the coefficients for the interpolating polynomial. The matrix V is the Vandermonde matrix with entries $V_{i,j} = x_i^{j-1}$.

$$V = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2^1 & x_2^2 & \cdots & x_2^{n-1} \\ 1 & x_3^1 & x_3^2 & \cdots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^1 & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}_{N \times N} \tag{2.1}$$

$$V\mathbf{a} = \mathbf{f} \tag{2.2}$$

It remains however to show that the interpolating function converges to the underlying function. For example, suppose equispaced samples are taken from a given

test function. If we then construct the interpolating polynomial, as we increase the number of samples, does the interpolating polynomial converge to the original function?

It turns out, mathematicians Carl Runge, C. Meray and Emilie Borel independently made the surprising discovery that the interpolating polynomial is actually likely to diverge! The phenomenon is now commonly known as Runge phenomenon. This consideration is one of the main motivations in developing the Minimum Sobolev Norm (MSN) interpolation scheme. Before going into the details, it will be more instructive to first look more closely at Runge phenomenon.

2.2 Runge Phenomenon

In this section, a completely intuitive explanation of the Runge phenomenon is presented. Carl Runge made the case that with polynomial interpolation on an equispaced grid where the number of coefficients is equal to the number of data points, the interpolating polynomial is likely to diverge with an increasing number of sample values. The function he used to show this is now commonly refered to as Runge's function. We take Runge's function in this explanation to be the family of functions which vary with the parameter α in equation 2.3.

$$f(x) = \frac{1}{1 + \alpha x^2},\ x \in [-1, 1],\ \alpha > 0 \tag{2.3}$$

In Figure 2.1, we show 11 samples of the Runge function, $\alpha = 10$, and the polynomial interpolant which use those samples. We then show in Figure 2.2, 19 samples of the same function, and its interpolating polynomial. The interpolation scheme here is the classical polynomial interpolation scheme described in Section 2.1. It can be seen that the interpolating polynomial is actually diverging with an increasing number of samples.

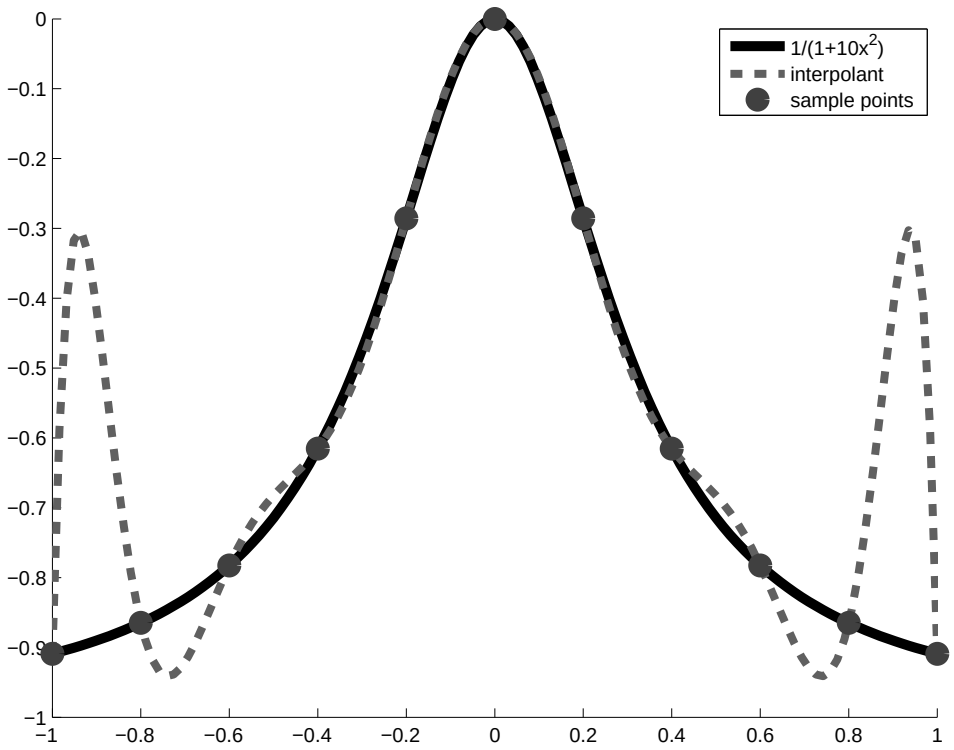

Figure 2.1: Runge phenomenon, 11 samples

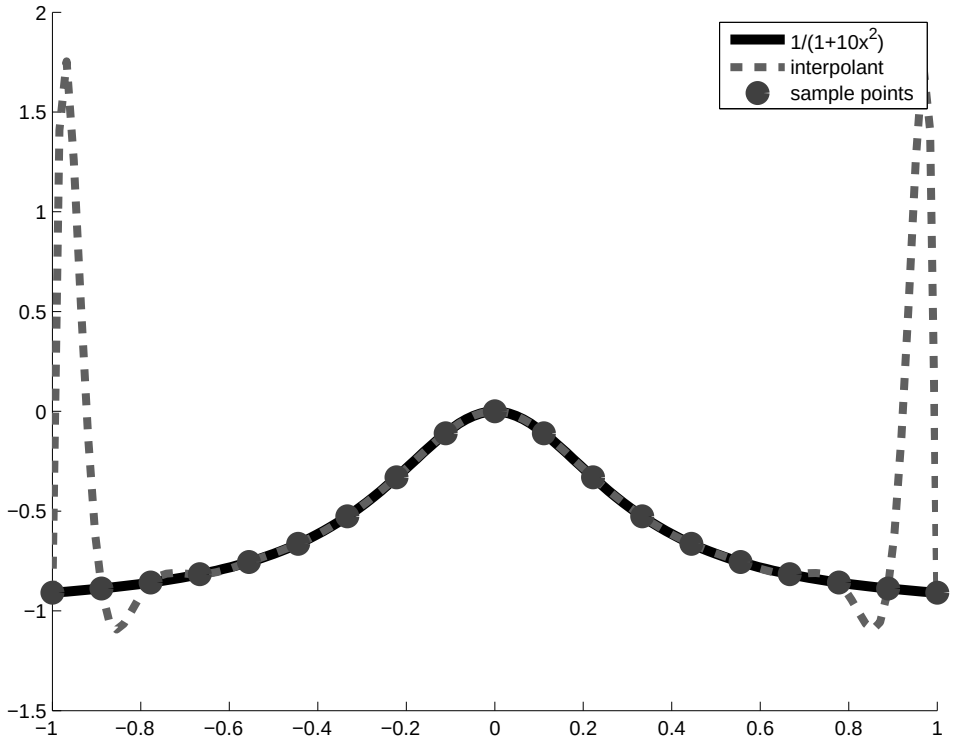

Figure 2.2: Runge phenomenon, 19 samples

This phenomenon arises due to the difference in shapes between Runge's function and the monomials, which form our basis. Runge's function dampens out to zero towards the boundaries, and the monomials all grow. Just this difference is not quite the whole story. It is the shape of the points in the middle region of each half of the axis that play the crucial part. By this middle region, we mean for example the region in $[-0.75, -0.25]$ and $[0.25, 0.75]$ when interpolation is being done on $[-1, 1]$. With Runge's function, these points can actually be controlled

by the parameter α. Increasing α makes the points in this middle region fall quicker, thus creating a sharper middle peak. By increasing and decreasing this parameter, we are shifting the two poles at $\pm\sqrt{\alpha}i$ closer and farther, respectively, to the origin.

Visually, to see how this causes Runge's phenomenon, first take the example of polynomial interpolation using 5 equispaced nodes on $[-1, 1]$. We will use as our underlying function a shifted Runge's function with parameter $\alpha = 1$. We shift Runge's function down by 1 just to aid in visualization. Before even looking at our interpolating polynomial, we know we will only have even powered coefficients, due to the symmetry of Runge's function. This example does not exhibit Runge's phenomenon just yet. The interpolating polynomial turns out to be $-0.9x^2 + 0.4x^4$, as shown in Figure 2.3

The x^2 component is required to bring the values of nodes at -0.5 and 0.5 down. The x^4 component was not large enough at -0.5 and 0.5 to accomplish this task, thus it is used to bring the two end points at -1 and 1 up. Interpolation is thus achieved. Of course, inverting the Vandermonde matrix to solve for the coefficients does not consider components one at a time in the way we just described. It is just a convenience that in this example we have a 'visually almost orthogonal' decomposition to work with.

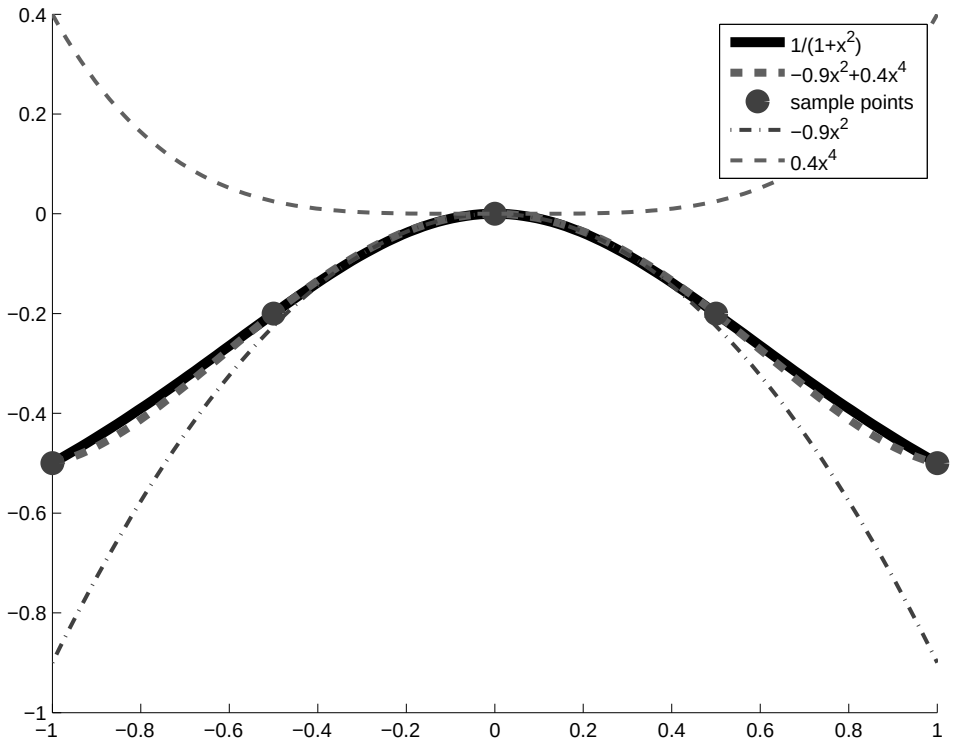

Figure 2.3: Runge phenomenon interpolant, 5 samples, $\alpha = 1$

Now, we raise the parameter α to be equal to 100. This brings down the two values at the midpoints -0.5 and 0.5 to equal almost -1 (recall we are using the shifted Runge function). This has the effect of making the x^2 component more negative. In effect, the x^4 component has to become larger in order for the end points to interpolate. The interpolating polynomial turns out to be $-4.8x^2+3.8x^4$, as shown in Figure 2.4.

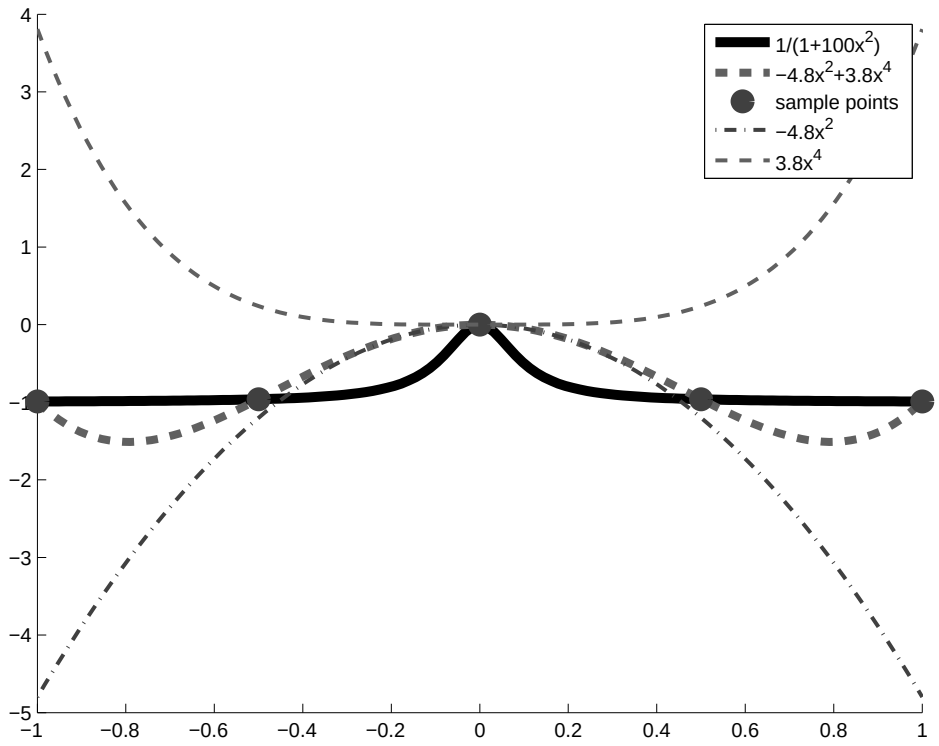

Figure 2.4: Runge phenomenon interpolant, 5 samples, $\alpha = 100$

Here is where Runge phenomenon makes its appearance. The function values of the nodes at $x = -1$ and $x = -0.5$ are essentially the same height. Thus, in order for the interpolating polynomial to go through those points, the x^2 component must fall the same amount the x^4 component rises in that interval. However, in that same interval, the x^4 component has a much greater curvature, thus a noticeable bubbling effect is seen in the resulting interpolant. This bubbling effect is the Runge phenomenon beginning to show itself.

In this discussion we have meant and still mean equispaced interpolation nodes. When we increase the number of nodes, more get evenly distributed along the region, but we also need a higher order polynomial to get a square system. Higher order monomials have an ever wider central flat region, and the quick increase near boundary sharpens. Thus, the bubbling effect gets worse with an increasing number of nodes as can be seen in the Figures 2.1 and 2.2.

One more minor point needs to be addressed. All during the previous discussion, we have only considered equispaced nodes. The location of the nodes have no bearing on the basis we are using. Having more nodes near the boundary where the higher order terms in the basis take a sharp climb actually enables us to avoid the Runge phenomenon. To visualize this, imagine the sharp increase of x^8. If we cluster more points near the boundary, we have sampled along the 'bubble', forcing that region to interpolate at the prescribed nodes. When done in the right way, this successfully avoids Runge phenomenon. The problem with this is that in most cases when using interpolation, the interpolation nodes are given to us. We often will not have a choice on the location of our data.

We have seen thus far that the amount of Runge phenomenon found in an interpolating polynomial is a result of the location of the underlying functions poles, and the location of interpolation nodes used. We can also view the effect from the perspective of the polynomial coefficients as frequency components of

the interpolating polynomial. Runge phenomenon is thus an effect where the high requenc components of the interpolant are growing. This can be seen from differentiating the Runge function and noting that the size of its derivatives are actually increasing in regions near the boundary. Looking at the problem in this light, we can intuitively develop an approach in the next section which suppresses the Runge phenomenon through a low pass filtering. This has to be done in a particular way to be effective, and we outline the approach in the next chapter on the Minimum Sobolev Norm (MSN) technique.

A mathematically rigorous understanding of the Runge phenomenon requires the evaluation of complex integrals using the theory of residues. The interested reader can find the details in [5].

Chapter 3

Minimum Sobolev Norm Interpolation

3.1 Intuition

Many attempts have been made at circumventing Runge phenomenon. This explains the vast number of interpolation schemes one finds in the literature. Splines, radial basis functions, conformal mappings, rational interpolants are all in a way an attempt to avoid or suppress Runge phenomenon.

In this section, a method that successfully suppresses the Runge phenomenon is described which we call the Minimum Sobolev Norm (MSN) interpolation scheme. It involves using a higher order polynomial compared to the number of interpolation nodes along with regularization constraints on the magnitude of the derivatives of the interpolant. Ideas similar to this have been previously explored. Fejer, the Hungarian mathematician, first proposed the idea of using a polynomial of

degree $2N$ where N is the number of interpolation nodes. The other half of the constraints were used to set the derivative at each node equal to zero in order to dampen out oscillations. With MSN, we take the idea a bit further and instead of controlling the derivative at each node, we control any derivative of our choosing across the whole interval. This is done by controlling the norm of the chosen derivative of the polynomial. The first step then is to find a norm which has information about the derivatives of a polynomial. We achieve this by using an appropriately defined Sobolev norm, and then set up an optimization problem which picks the interpolating polynomial for which this norm is minimum.

The following discussion and development will take a route more guided by intuition as opposed to a rigorous mathematical approach. Since the purpose of this thesis is to communicate these ideas in a concise and intuitive way to working engineers, the more rigorous details along with the proofs will be left out. The interested reader can find these in [3].

3.2 Size of Fourier Series Coefficients

Information about the smoothness of a function is directly related to the rate of decay of its Fourier series coefficients. This is related to the summability of the Fourier coefficients. Consider the Fourier series coefficients of an absolutely

continuous function f, where $\hat{f}(m)$ denotes the n-th Fourier coefficient. Then,

$$|\hat{f}(m)| \leq \frac{K}{|m|}. \tag{3.1}$$

Thus if a periodic function on $[0, 2\pi]$ is s times differentiable and its s-th derivative is continuous, then the rate of decay of its Fourier coefficients is faster than $\frac{1}{m^s}$. To see why, consider differentiating the Fourier series. Each coefficient $\hat{f}(m)$ will be weighted by an $(im)^s$, thus for the functions s-th derivative to be absolutely continuous, the coefficients before differentiation must be decaying at least at the rate $\frac{1}{m^s}$. In engineering terms, differentiation boosts high frequency components.

Thus, if we require a periodic function have 3 continuous derivatives, then its Fourier coefficients must decay by at least $\frac{1}{m^3}$. We enforce a similar constraint when we solve for the MSN interpolant. In the MSN approach, we use a much higher degree polynomial than the number of nodes, thus (as with ordinary polynomial interpolation) there are an infinite number of solutions for the coefficients. As the extra constraint we weight each coefficient a_m by m^3 and ask that the resulting 2-norm of these weighted coefficients be minimum. The approach thus controls the rate of decay of the coefficients, thus controlling the resulting smoothness. Since we are using a 2-norm minimization, we are in a sense asking that the 3rd derivative (in this example) is controlled on average (in the 2-norm sense) over the whole interval. Ultimately, the desired effect of removing Runge phenomenon while maintaining interpolation is achieved. This method has provable

convergence, and many of the ideas fall into the framework of Sobolev spaces. We take a short look at some of the essential ideas guided by intuition in the following discussion.

3.3 Picking a Sobolev Norm

A Sobolev norm is a norm which measures the magnitude of the derivatives of a function. More specifically, we are want a measure of the magnitude of the derivatives across the whole interval of interest, in an average sense. This is opposed to a pointwise measure, such as getting a measure of the maximum value that a derivative takes. Thus, we seek a 2-norm measure. We want to find such a measure which works for polynomials, denoted as p.

As it will turn out, we do not require an exact measure of the derivatives of the polynomial p. The 2-norm nature of the Sobolev norm we seek allows us some extra freedom from exactness. To get our Sobolev norm, we measure the magnitude of the derivatives of a closely related function instead. We choose the Chebyshev polynomials as our basis for p (instead of the usual monomials). The Chebyshev polynomials are defined by the recurrence relation in equation 3.2 or by the trigonometric formula in equation 3.3. The recurrence relation formulation

is more stable when evaluating derivatives at the boundary, but essentially both are equivalent.

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) + T_{n-1}(x) \end{aligned} \tag{3.2}$$

$$T_n(x) = \cos(n \arccos(x)),\ n = 0, 1, 2... \tag{3.3}$$

Denote the M-th order Chebyshev expansion of p as $p_M(x)$ shown in equation 3.4. In this basis, there is a simple path to obtaining a Sobolev norm of the closely related function $(p \circ g)(\theta)$ where $g(\theta) = cos(\theta)$ and $p = p_M(x)$.

$$p_M(x) = \sum_{m=0}^{M-1} a_m T_m(x) \tag{3.4}$$

$$p_M(x) = \sum_{m=0}^{M-1} a_m \cos(m \arccos x) \tag{3.5}$$

$$\Rightarrow p_M(\cos\theta) = \sum_{m=0}^{M-1} a_m \cos m\theta \tag{3.6}$$

A quick visualization of the function composition $(p \circ \cos)(\theta)$ is in order. Moving over the region from π to 2π in θ corresponds to moving from -1 to 1 in x. In addition, moving from π to 0 in θ corresponds also to moving from -1 to 1 in x. Thus, $(p \circ \cos)(\theta)$ is an even 2π periodic function. More simply put, with

each Chebyshev polynomial term, we are just creating a cosine from $[0, 2\pi]$. Thus, our new function $(p \circ \cos)(\theta)$ is a cosine series using the coefficients $\mathbf{a}$ from our Chebyshev expansion of p.

Since it is well understood how the decay of the Fourier series of a function is related to its smoothness, we define the Sobolev norm on this new composed function $(p \circ \cos)$. This is straightforward to do since the only thing we have to differentiate are sinusoids. The Sobolev norm simply becomes the weighted 2-norm of our coefficients a_m, with the weights m^s. We take cosines to have unit norm. The constant s which represents the s-th derivate is now a parameter. It tells us which derivative we have used in calculating the Sobolev norm.

$$\begin{aligned} \|p_M(\theta)\|_s^2 &= \|\frac{d^s}{d\theta^s} p_M(\theta)\|_2^2 && (3.7) \\ &= \sum_{m=0}^{M-1} |a_m|^2 m^{2s} && (3.8) \\ &\approx \|D_s \mathbf{a}\|_2^2 && (3.9) \end{aligned}$$

where the matrix D_s in the above equation is defined as

$$D_s = \begin{bmatrix} 1^s & 0 & ... & 0 & 0 \\ 0 & 2^s & ... & 0 & 0 \\ . & . & ... & . & . \\ 0 & 0 & ... & 0 & M^s \end{bmatrix}_{M \times M} . \tag{3.10}$$

We now have a measure of magnitude of the s-th derivative of ($p \circ \cos$), through a weighted 2-norm calculation. This Sobolev norm is in the average (weighted 2-norm) sense, and since it is applied spectrally in θ, has meaning over the whole interval. It should be apparent now that if we setup an underdetermined interpolation problem on the Chebyshev polynomials from which we choose coefficients whose Sobolev norm with parameter s (as defined in equation 3.10) is minimized, we have chosen the interpolant which has the smallest energy in the s-th derivative, in an average sense over the whole interval, out of the infinite choices of polynomials.

Note we have defined the Sobolev norm spectrally on ($p \circ \cos$) instead of directly on $p_M(x)$. Thus one can raise the following question. We were able to define a Sobolev norm for ($p \circ \cos$), but what does this say about $p_M(x)$ (the function which we are interested in) and its derivatives? Is it a reliable measure of the derivatives of $p_M(x)$? Take as an example $p_M(x) = x$, then $(p \circ \cos) = cos(\theta)$. We now look at the first derivative of each. $p'_M(x)$ is equal to 1 along the whole interval $[-1, 1]$,

however on $[\pi, 2\pi]$, $(p \circ \cos)' = -sin(\theta)$. Starting at π and moving towards 2π, $(p \circ \cos)'$ starts at 0, moves to 1, and ends up at 0 again. Pointwise, there are sharp distortions between the two. However, looking on average, the derivatives are mostly the same across the two intervals. This agrees intuitively with our approach of using the weighted 2-norm spectrally in obtaining the Sobolev norm, since as we know from Parsevals identity the 2-norm applies equally in the spatial and spectral domains. Looking more closely, the connection above between $(p \circ \cos)'$ and $p'_M(x)$ includes a $\frac{1}{\sqrt{1-x^2}}$ term. This term is integrable, even though pointwise it blows up at the boundaries, thus in an average sense, the connection is meaningful, though the exact analysis is subtle.

Ultimately, however, the nature of this connection is irrelevant. The key point, as we will see, is that weighting down the Chebyshev coefficients with D_s produces converging interpolants free from Runge phenomenon, regardless of how we interpret it. We do the weighting spectrally out of simple numerical practicality, due to the fact that it becomes a diagonal matrix multiplication, and in addition, is more well conditioned as compared to applying the same operator in the spatial domain. Interpretations about what it exactly means mathematically for $p_M(x)$ and its derivatives is not important. The fact is, it works! Proofs of convergence can be found in [3], and do not hinge on any connection between the derivatives of $(p \circ \cos)$ and $p_M(x)$, the discussion here is given only to satisfy our intuition.

3.4 The MSN solution

As we have pointed out, the Sobolev norm we have defined is a sufficient measure (in an average sense over the interval) of the s-th derivative of a polynomial. In addition, the norm can be written as a diagonal matrix multiplication if we know the actual Chebyshev coefficients. Returning to the problem of interpolation, if we choose an interpolating polynomial whos order is much larger than the number of interpolation nodes, we end up with underdetermined system and we must use some additional constraints to choose one of the infinite number of solutions. We use the above Sobolev norm as the additional constraint, and ask that it be minimum. Using s=2 for example, and a polynomial degree M, we can interpret that the coefficients solved for correspond to the M-th degree polynomial which in the sense of our Sobolev norm, has smallest second derivative across the whole interval.

With this idea in place, we are ready to state the minimization problem of MSN interpolation. The idea is to find the interpolating polynomial defined by the coefficients $\mathbf{a}$ for which the s-th Sobolev norm as described in equation 3.8 is minimum.

$$\mathbf{a} = \arg \min_{\mathbf{a}: V\mathbf{a}=\mathbf{f}} \|D_s \mathbf{a}\|_2^2 \tag{3.11}$$

We will present later a precise way of choosing the order M, but assume now that M is larger than N. For interpolation in 1d, a good choice is $M = 3N$.

$$V\mathbf{a} = \mathbf{f} \tag{3.12}$$

Equation 3.12, is an underdetermined system, thus there are infinitely many solutions. In order to choose the one with minimum Sobolev norm, we multiply the coefficients to be solved for with the Sobolev weight matrix. In order to keep the same equations, we multiply its inverse on the left. Note D_s as defined in equation 3.10 is invertible.

$$\Rightarrow VD_s^{-1}D_s\mathbf{a} = \mathbf{f} \tag{3.13}$$

We now simply treat $D_s\mathbf{a}$ as a new unknown $\mathbf{z}$ for which we are solving for.

$$\Rightarrow VD_s^{-1}\mathbf{z} = \mathbf{f} \tag{3.14}$$

We can now solve for the minimum norm solution $\mathbf{z}$ using the pseudo-inverse.

$$\mathbf{z} = (VD_s^{-1})^{\dagger}\mathbf{f} \tag{3.15}$$

$$\Rightarrow \mathbf{a} = D_s^{-1}(VD_s^{-1})^{\dagger}\mathbf{f} \tag{3.16}$$

$$= D_s^{-2}V^T(VD_s^{-2}V^T)^{-1}\mathbf{f}. \tag{3.17}$$

If $\mathbf{z}$ has minimum norm, so does $D_s\mathbf{a}$. Thus, we have in fact chosen the coefficients $\mathbf{a}$ with minimum Sobolev norm.

The interpolant at some $x \in [-1, 1]$ is then given by the sum

$$\begin{aligned} p_M(x) &= \sum_{m=0}^{M-1} a_M T_m(x) \\ &= V(x)\mathbf{a}, \end{aligned} \tag{3.18}$$

where $V(x) = [\ T_0(x) \quad T_1(x) \quad ... \quad T_{M-1}(x) \quad T_{M-1}(x)\ ]$.

The choice of the order M and the Sobolev weight parameter s play a crucial role in determining whether the interpolant will converge or diverge. Once appropriately chosen, convergence in N, the number of points, can be achieved. Convergence results can be found in [2] and proofs in [3].

Intuitively, the order M should be chosen based on the smallest grid spacing of the points N. This makes sense, since two very close points which jump largely in their data value may require a much higher degree polynomial for interpolation with the additional constraint for Runge suppression. We define the meshnorm $I(x_N)$ as the following:

$$I(x_N) = \left\lceil \frac{\pi}{\min_{i \neq j} \|\theta_i - \theta_j\|} \right\rceil, \tag{3.19}$$

where $\theta_i = \cos^{-1} x_i$. We require that $M = cI(x_N)$ for convergence. In the case of equispaced gridding, we can safely set the order $M = 3N$.

The choice of s depends on the smoothness of the underlying function of which we are trying to approximate. For very rough functions, such as $|x|$, s can be

chosen as low as 0.5. For functions with 2 continuous derivatives, we require that s be a minimum of 2. Note, s need not be an integer.

Finally, note that we do not achieve convergence in M or in s. For a fixed N, increasing s may seem to increase the accuracy of the interpolant for a while, but ultimately does not achieve convergence. This can be seen intuitively, if we have only 5 points, no matter how high we choose s, we are not gaining anymore information about our underlying function. A similar argument follows for M.

3.5 MSN in higher dimensions

In 2d, our grid points now become the vector $\mathbf{x}_i = \{x_i, y_i\}_{i=0}^N$. The corresponding meshnorm is now

$$I(\mathbf{x}_i) = \left\lceil \frac{\pi}{\min_{i \neq j} \|\theta_i - \theta_j\|_2} \right\rceil, \tag{3.20}$$

where $\theta_i = \{\cos^{-1} x_i, \cos^{-1} y_i\}$. In practice a safe estimate for the order is $M = \frac{4}{\lceil \min_{i \neq j} \|\theta_i - \theta_j\|_2 \rceil}$.

For completeness, we also show how the Chebyshev vandermonde matrix along with the appropriate Sobolev weights can be formed.

$$\mathbf{T}_{m,n}(\mathbf{x}) = T_m(x) T_n(y),\ 0 \leq m < M_x,\ 0 \leq n < M_y, \tag{3.21}$$

$$V = \begin{bmatrix} \mathbf{T}_{0,0}(\mathbf{x}_0) & \mathbf{T}_{0,1}(\mathbf{x}_0) & ... & \mathbf{T}_{0,M_y-1}(\mathbf{x}_0) & ... & \mathbf{T}_{M_x-1,M_y-1}(\mathbf{x}_0) \\ \mathbf{T}_{0,0}(\mathbf{x}_1) & \mathbf{T}_{0,1}(\mathbf{x}_1) & ... & \mathbf{T}_{0,M_y-1}(\mathbf{x}_1) & ... & \mathbf{T}_{M_x-1,M_y-1}(\mathbf{x}_1) \\ . & . & . & . & . & . \\ \mathbf{T}_{0,0}(\mathbf{x}_{N-1}) & \mathbf{T}_{0,1}(\mathbf{x}_{N-1}) & ... & \mathbf{T}_{0,M_y-1}(\mathbf{x}_{N-1}) & ... & \mathbf{T}_{M_x-1,M_y-1}(\mathbf{x}_{N-1}) \end{bmatrix} \tag{3.22}$$

The Sobolev weight matrix D_s can be defined in the following way.

$$D_s = \begin{bmatrix} (1+0^2+0^2)^{\frac{s}{2}} & 0 & 0 & ... & 0 \\ 0 & (1+0^2+1^2)^{\frac{s}{2}} & 0 & ... & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & ... & (1+(M_x-1)^2+(M_y-1)^2)^{\frac{s}{2}} \end{bmatrix} \tag{3.23}$$

Chapter 4

Full MSN (FMSN)

4.1 Introduction

We have shown how the MSN idea can be used to solve for an interpolating polynomial which is free from Runge phenomenon. Now we extend the idea to the numerical solution of PDEs. Those readers unfamiliar with the topic of PDE solvers can look to the appendix for a more detailed explanation and development with simple examples applied to ODEs. There, we first solve globally a 1d boundary value problem using FMSN, present some numerical and memory issues, then show how solving the same problem with the local FMSN approach remedies those issues. The complete method which we propose as a PDE solver uses the local FMSN approach, and is explained with sufficient detail in the following section.

4.2 FMSN Solution of PDEs

In this section, we present a method which numerically approximates the solution to PDEs such that the global solution obtained is of minimum Sobolev norm. What we mean by this will be explained shortly. Assume the PDE operator is given by $\mathcal{L}$ and the boundary equation is given by $\mathcal{B}$. The PDE we wish to solve is given generally as:

$$\mathcal{L}(u(x,y)) = f(x,y), (x,y) \in \Omega \tag{4.1}$$

$$\mathcal{B}(u(x,y)) = g(x,y), (x,y) \in \partial\Omega. \tag{4.2}$$

The coordinates for the discretization of the interior region Ω and the boundary $\partial\Omega$ are stored in the vectors $\mathbf{x}$ and $\mathbf{y}$ to form the list of coordinates $(\mathbf{x}, \mathbf{y})$. The total number of discretization points from the boundary and interior total to N. We can split these into interior points and boundary points using the following notation. The interior region is discretized with N_I points and is denoted $(x_j^{int}, y_j^{int}), j = 1, 2, ...N_I$. The boundary is discretized with N_B points $(x_j^{bnd}, y_j^{bnd}), j = 1, 2, ...N_B$. These are the discretization nodes for which we can specify knowns and unknowns, depending on the PDE and boundary conditions. In the example PDEs which we solve in this thesis, the values of $f(x,y)$ are known at the interior points $(\mathbf{x^{int}}, \mathbf{y^{int}})$ and are stored in $\mathbf{f}$. The boundary values are known points in $(\mathbf{x^{bnd}}, \mathbf{y^{bnd}})$ and

are stored in $\mathbf{g}$. We will be solving for the unknowns $\mathbf{u}$ at the interior points $(\mathbf{x^{int}}, \mathbf{y^{int}})$.

We place specification points contained in $(\mathbf{c_x}, \mathbf{c_y})$ which are the centers (c_{xi}, c_{y_i}), $l = 1, 2...k$ of all the windows to be placed down. A window is used to select local discretization points in $(\mathbf{x}, \mathbf{y})$. The set of points which falls under each window is denoted by the set C_l, defined in equation 4.3. Assuming the interior grid $(\mathbf{x^{int}}, \mathbf{y^{int}})$ is rectangularly equispaced, let h_h denote the horizontal grid spacing and h_v denote the vertical grid spacing. The window size can be setup as a rectangular window, based on $L-1$, the number of grid spacings a window will cover in each direction.

$$C_l = \{(x_n, y_n) : \|x_n - c_{xl}\|_\infty \leq \frac{h_h(L-1)}{2} \cap \|y_n - c_{y_l}\|_\infty \leq \frac{h_v(L-1)}{2}\}, l = 1, 2...k \tag{4.3}$$

Choosing a sufficient number of specification points and window size is the next task. We conjecture that for n-th order PDEs, the windows need at least n overlapping points along any direction. We assume in the discussion this criteria is met. Window placement will be covered more in depth in the following sections on implementation. Assuming we have properly placed our k windows, our aim is to solve the following minimization problem, where each i represents a window centered at some point (c_{xi}, c_{y_i}).

$$\min_{\forall i, B_i \mathbf{a}_i = \mathbf{z}_i} \sum_{i=1}^{k} \|D_s \mathbf{a_i}\|_2^2 \tag{4.4}$$

Each i represents a different window. In general, $B_i \mathbf{a}_i = \mathbf{z}_i$ is the following equation,

$$\begin{bmatrix} \mathcal{L}V(\phi(\mathbf{x}_i^{int})) \\ V(\phi(\mathbf{x}_i^{int})) \\ \mathcal{B}V(\phi(\mathbf{x}_i^{bnd})) \end{bmatrix} \mathbf{a} = \begin{bmatrix} \mathbf{f}_i^{int} \\ \mathbf{u}_i^{int} \\ \mathbf{g}_i^{bnd} \end{bmatrix} \tag{4.5}$$

where $\mathbf{f}_i^{int}$ and $\mathbf{b}_i^{bnd}$ are the known values of $\mathbf{f}$ and $\mathbf{g}$ which fall under the i-th local window. $\mathbf{u}_i^{int}$ are the local unknowns on the interior which fall under the i-th window. In the examples we solve in this paper, all the unknowns fall are assumed to be on the interior, but this need not be the case. We also require an affine mapping $\phi(x)$ to map the points which fall under the window C_i to be within the range $[-1, 1]$, as the Chebyshev polynomials are only defined on $[-1, 1]$.

The first observation to make is that the minimizations in equation 4.4 are over each local set of coefficients $\mathbf{a}_i$. We have stated that $\mathbf{u}$ is an unknown in the PDE which we are trying to solve, but at this point in the discussion it is only part of the constraint in each minimization. Furthermore, since each window must overlap, each minimization in the summation is coupled through $\mathbf{u}$.

Let us assume now that we have some $\mathbf{u}$ which satisfies every constraint equation, thus $\forall i$, $B_i \mathbf{a}_i = \mathbf{z}_i$. We now have an infinite choice over what $\mathbf{a}_i$ to choose

to try and satisfy the global minimum. What we can do is get a formula for the optimal $\mathbf{a}_i$ that works for such a $\mathbf{u}$, that is in terms of $\mathbf{u}_i$. We can use this to plug back into each minimization, thus rendering the minimizations over $\mathbf{u}_i$ only. Thus what remains is to find each $\mathbf{u}_i$ which makes the global optimization problem minimum. But this is much simpler since the $\mathbf{u}$ of which we can choose from have already met the constraints. Thus, the problem becomes a standard unconstrained least squares minimization problem. In order to get the actual formula for the optimal $\mathbf{a}_i$, we do some linear algebra to get each $D_s\mathbf{a}_i$ in terms of each $\mathbf{z}_i$ (which contains $\mathbf{u}_i$), to then be plugged back inside the minimization. Take a single minimization, indexed by i,

$$\min_{B_i\mathbf{a}_i=\mathbf{z}_i} \sum_{i=1}^{k} \|D_s\mathbf{a_i}\|_2^2 \tag{4.6}$$

Assuming that we have some $\mathbf{u}_i$, we can express the coefficients $\mathbf{a}_i$ in terms of $\mathbf{z}_i$ through a pseudoinverse of B_i (recall $\mathbf{z}_i$ contains $\mathbf{u}_i$). Numerically, this can be achieved through a QRV factorization. D_s is chosen to have the appropriate Sobolev weights as described in equations 3.23. In our experiments, we have used what is known as the 'economy' SVD for this QRV factorization, where V^T is m by n (contains only its first m rows) and Σ is m by m, given m<n.

First, insert $D_s^{-1}D_s$ into the constraint equation,

$$B_i \mathbf{a}_i = \mathbf{z}_i \tag{4.7}$$

$$B_i D_s^{-1} D_s \mathbf{a}_i = \mathbf{z}_i \tag{4.8}$$

and form the pseudoinverse taking the SVD of $B_i D_s^{-1}$.

$$U_i \Sigma_i W_i^T = B_i D_s^{-1} \tag{4.9}$$

$$U_i \Sigma_i W_i^T D_s \mathbf{a}_i = \mathbf{z}_i \tag{4.10}$$

$$W_i^T D_s \mathbf{a}_i = \Sigma_i^{-1} U_i^T \mathbf{z}_i \tag{4.11}$$

Since W_i is orthogonal, and does not affect the norm, we can plug this back into the minimization:

$$\begin{aligned} \min_{\mathbf{u}_i, \mathbf{a}_i} \|W_i^T D_s \mathbf{a}_i\|_2^2 &= \min_{\mathbf{a}_i} \|D_s \mathbf{a}_i\|_2^2 & (4.12) \\ &= \min_{\mathbf{u}_i} \|\Sigma_i^{-1} U_i^T \mathbf{z}_i\|_2^2 & (4.13) \end{aligned}$$

Note the final minimization in equation 4.13 is over $\mathbf{u}_i$ only. Thus we have removed the coefficients $\mathbf{a}_i$ from the minimization, but have kept an equivalent minimization problem. Note, at this point the minimizations are still coupled, due to the overlapping of windows, but now, we only have to work with $\mathbf{u}$. Thus, we have an unconstrained minimization, which is a sum of squared 2-norms, each of which are linear in each $\mathbf{u}_i$. We call the matrix $\Sigma_i^{-1} U_i^T$ as M_i and partition its columns

based on the rows which were used for B_i and z_i in equation 4.5.

$$\Sigma_i^{-1} U_i^T = M_i = \begin{bmatrix} M_{f_i} & M_{ui} & M_{bi} \end{bmatrix} \tag{4.14}$$

The global minimization in equation 4.4 can now be expressed solely in terms of $\mathbf{u}$ as

$$\min_{\mathbf{u}_i} \sum_{i=1}^{k} \left\| M_i \begin{bmatrix} \mathbf{f}_i \\ \mathbf{u}_i \\ \mathbf{g}_i \end{bmatrix} \right\|_2^2 \tag{4.15}$$

where each minimization in the sum can be written as

$$\min_{\mathbf{u}_i} \left\| \begin{bmatrix} M_{f_i} & M_{ui} & M_{bi} \end{bmatrix} \begin{bmatrix} \mathbf{f}_i \\ \mathbf{u}_i \\ \mathbf{g}_i \end{bmatrix} \right\|_2^2. \tag{4.16}$$

Since each local minimization is over $\mathbf{u}_i$ we can rewrite the above equation as

$$\min_{\mathbf{u}_i} \sum_{i=1}^{k} \left\| M_{ui}\mathbf{u}_i - (-M_{f_i}\mathbf{f}_i - M_{bi}\mathbf{g}_i) \right\|_2^2. \tag{4.17}$$

Thus, each local minimization is a standard least squares minimization. We still can not proceed by separately minimizing each of these terms, which are in the global minimization of equation 4.15, since they are coupled by overlapping windows. However, we can combine the sum of the above least squares minimiza-

tions into a single least squares matrix minimization problem, over the full $\mathbf{u}$. Keeping columns consistent for each point in $\mathbf{f}$,$\mathbf{u}$, and $\mathbf{g}$, we can form the grand sparse matrices M_{Gu},M_{Gf}, and M_{Gb}, with a set of rows for each of the individual least squares minimization problems in equation 4.16. We can setup the following minimization which solves our PDE for $\mathbf{u}$ and obtains the global minimization of equation 4.13.

$$\min_{\mathbf{u}} \left\| M_{Gu}\mathbf{u} - (-M_{Gf}\mathbf{f} - M_{Gb}\mathbf{g}) \right\|_2^2 \tag{4.18}$$

$$M_{Gu}\mathbf{u} \overset{LS}{=} -M_{Gf}\mathbf{f} - M_{Gb}\mathbf{g} \tag{4.19}$$

Note, proper indexing must be used to form the grand sparse matrices M_{Gu},M_{Gf}, and M_{Gb}. That topic is covered in some more detail in the following section.

The generality of the approach is where this method derives its power. We have a flexible control over memory use and computation time by working with the window sizes and locations. The local approach also lends itself to setting up a variety of PDEs. Most importantly, the method is a least squares approach. The method is still in its early stages, but we see advantage in not requiring a square system. In cases where the number of equations required for a solution to the PDE is not exactly clear, for example with div-curl problems, the least squares system could prove quite valuable.

4.3 Sparse Matrix Indexing

In this section, we go into some more detail regarding forming the grand sparse matrices M_{Gu},M_{Gf}, and M_{Gb}.

When setting up the local matrix B_i, the discretization points in $(\mathbf{x}, \mathbf{y})$ which fall under each window indicate the equations which are associated with each points (PDE on interior points, boundary equation on the boundary points). If a window falls in the center region and contains only interior points, then the bottom block row $\mathcal{B}V(\phi(\mathbf{x}_i^{bnd}))$ of B_i (as well as the rows of $\mathbf{g}_i^{bnd}$ in $\mathbf{z}_i$) will not exist. Consequently, there will be no local matrix M_{bi} in equation 4.16 for that window. Thus, when we solve for each local M_i in equation 4.14, the grid points which formed each row of B_i and $\mathbf{z}_i$ are the same grid points corresponding to the columns of matrices M_{f_i},M_{ui} and M_{bi} since we have essentially taken a psuedoinverse of the matrix B_i.

These locally solved for matrices M_{f_i},M_{ui}, and M_{bi} must then be placed in the grand sparse matrices M_{Gu},M_{Gf}, and M_{Gb}. These matrices contain the matrices M_i from each local solve, and are thus global matrices. The grand sparse matrices M_{Gu} and M_{Gf} each have N_I columns corresponding to all the interior points of $(\mathbf{x}, \mathbf{y})$. M_{Gb} has N_B columns corresponding to every point on the boundary. Thus, we emphasize the importance of having an index for every point in $(\mathbf{x}, \mathbf{y})$ to be

able to used for indexing specific columns of the grand matrices M_{Gf},M_{Gu}, and M_{Gb}.

Each local solve for M_i gets its own set of rows in the grand matrices. Thus, if for example the i=1 window which was used in solving for M_1 had 10 interior points, and 4 boundary points, the block row matrices of B_1 given by $\mathcal{L}V(\phi(\mathbf{x}_1^{int})), V(\phi(\mathbf{x}_1^{int})), \mathcal{B}V(\phi(\mathbf{x}_1^{bnd}))$ would have 10, 10, and 5 rows, respectively. Thus, M_1 is a square matrix with 25 rows and columns. Thus, 25 rows are set apart in the grand matrices M_{Gu},M_{Gf}, and M_{Gb} for which the column blocks $\left[M_{f_1}\right]_{25\times 10}$,$[M_{u1}]_{25\times 10}$, and $[M_{b1}]_{25\times 5}$ (all from M_1) will be placed.

The local matrices M_{f_1},M_{u1}, and M_{b1} will fill in the exact columns of the grand matrices which correspond to the same points which went into the rows of B_1 and $\mathbf{z}_1$. This emphasizes the point of setting up a consistent indexing to enable the filling in of the global matrices M_{Gf},M_{Gu}, and M_{Gb}. Thus, each window i will have its own set of rows, but will share columns, depending on how much windows overlap.

Once we have filled in the grand matrices M_{Gf},M_{Gu}, and M_{Gb} after completing all the local solves for M_i, we can proceed to solving the global optimization problem in equation 4.18 by means of equation 4.19.

4.4 Window Placement

We will run though some examples of placing windows, varying the overlap, and noting its effect on accuracy. All of our examples will have $\mathbf{x}$ be on the square domain $(x, y) \in [-0.5, 0.5], [-0.5, 0.5]$ with equispaced gridding, corners removed.

Our first option is placing a window at every point in $\mathbf{x}$. The solution attains the highest accuracy due to the consistent bandwidth of M_{Gu} being close to 25, using and $L = 5$ window. The window on the interior is selecting all the points it should (25 points) as an $L = 5$ window. However, the matrix M_{Gu} is the largest, and computation time is the longest. In this case, we say the overlap parameter O was chosen to be 3, since two $L = 5$ windows placed on adjacent points on a equispaced grid will overlap over the distance of $3h$ units, with 4 grid points on top of each other.

The second option is coming up with a scheme which places windows in an equispaced manner, but not at every point in $\mathbf{x}$. To do this, given a fixed window size L, we then choose an overlap amount. Placing windows down on the square region from $[a, b]$, for a given L, we can find the number of windows which fit into $[a, b]$ with an overlap O of our choosing as:

$$W_{width} = (L-1)h \tag{4.20}$$

$$S = W_{width} - Oh \tag{4.21}$$

$$N_w = 2\lfloor (b-a)/(2S) \rfloor + 1 \tag{4.22}$$

W_{width} represents the total width of the window; a window when placed on a sample which can select $L = 5$ equispaced samples (that are h apart) is $4h$ wide. S denotes the spacing of the center of the windows, after an overlap of O has been accounted for. N_w is the number of windows which fit into the region $[a, b]$. When a window is placed at all the N_w centers, we are not guaranteed to cover every point in the region, since we have floored the result for N_w.Basically, a case could arise when N_w windows is an insufficient covering, but adding one more window on each side S away means those two additional windows fall outside the region $[a, b]$. We can test for this case, and when it occurs (for some instances of L and O) we just place additional windows on the boundary instead. In 2d, these extra boundary windows will use the same spacing S along the boundary. We show some examples of what we mean in the following discussion. The 2d region we consider is the square region $(x, y) \in [-0.5, 0.5], [-0.5, 0.5]$.

Shown below is an overlap $O = 3$ with window size $L = 5$. This time however, the windows are placed between points, instead of on top of points as in the first

case we discussed. Note, both the overlap amount O and the window size L are in relation to the equispaced gridding of $\mathbf{x}$. Two adjacent windows in 2d are shown in Figure 4.4. Boundary points are diamonds, interior points are circles, and window centers are filled in diamonds.

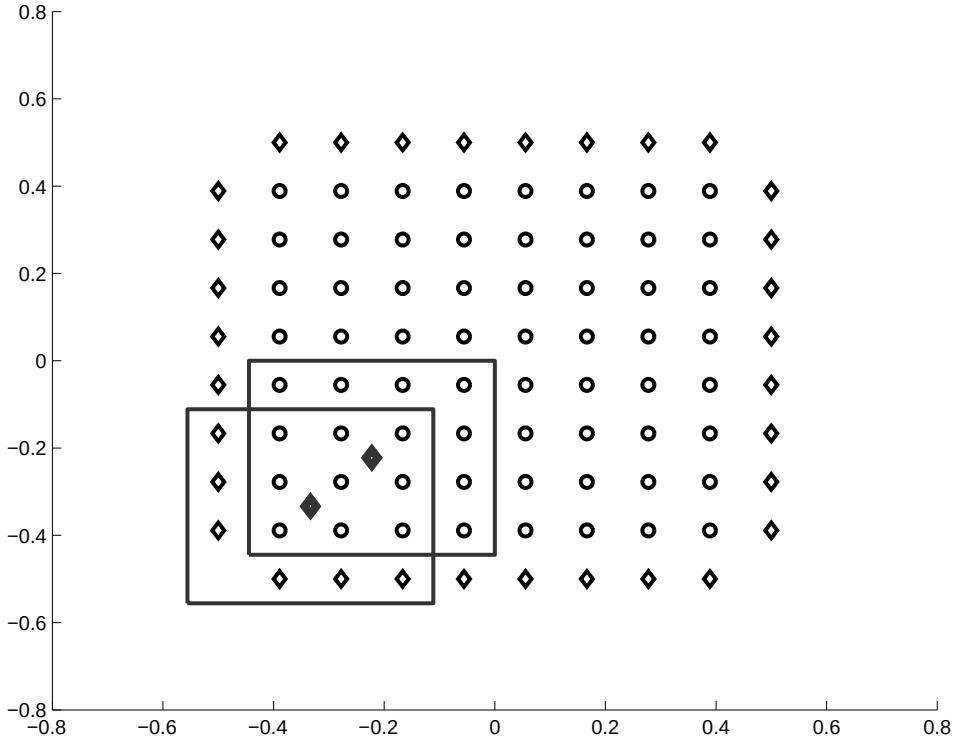

Figure 4.1: Two adjacient windows of size L=5 and overlap O=3

An overlap of 2 guarantees that regardless of window placement, each window will overlap at least two points in any direction. See Figure 4.4 for two adja-

cent windows. Again, boundary points are diamonds, interior points are circles, window centers are filled in diamonds.

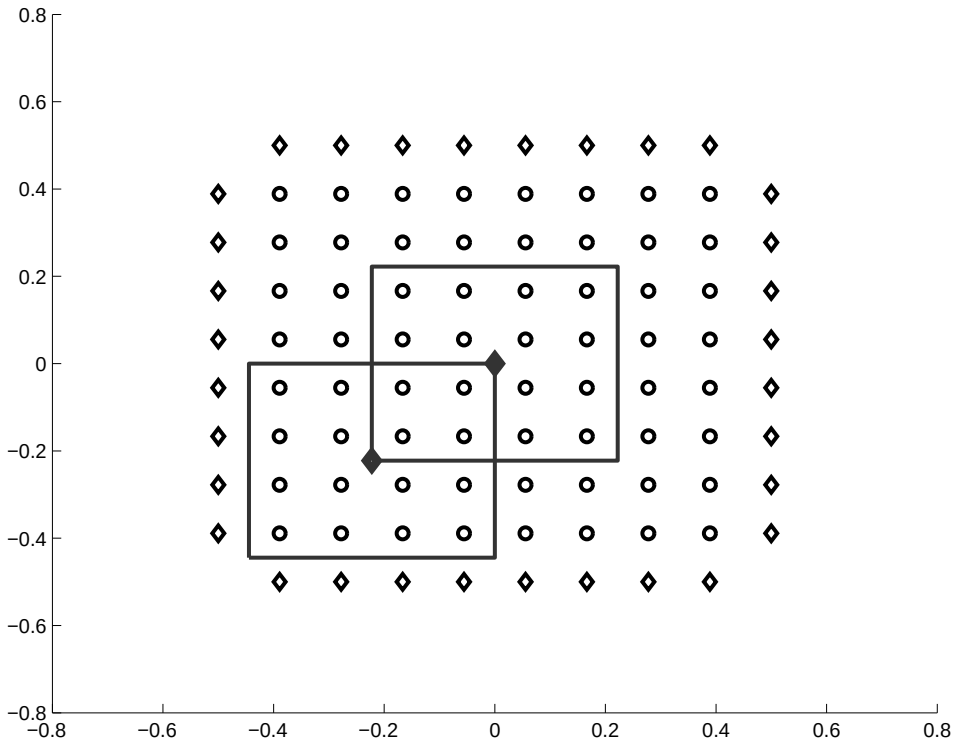

Figure 4.2: Two adjacent windows of size L=5 and overlap O=2

An even more optimal placement is available when we choose an O of 1.5. Care must be taken in using an overlap of less than the amount required by the PDE order. Assume for the discussion we solving a second order PDE. Notice that with the location shown below of the $L = 5$ windows, an overlap of 1.5 still guarantees that two points always fall under the overlapping sections of the

window. Here we have found an optimal spacing for equispaced grids for second order problems. The accuracy of the solution under this gridding remains since we have overlapped at 2 points with every window, and the number of rows in the grand matrix Mu is minimized. Figure 4.4 shows a few chosen windows along this grid. Some windows are left out in order to aid visualization.

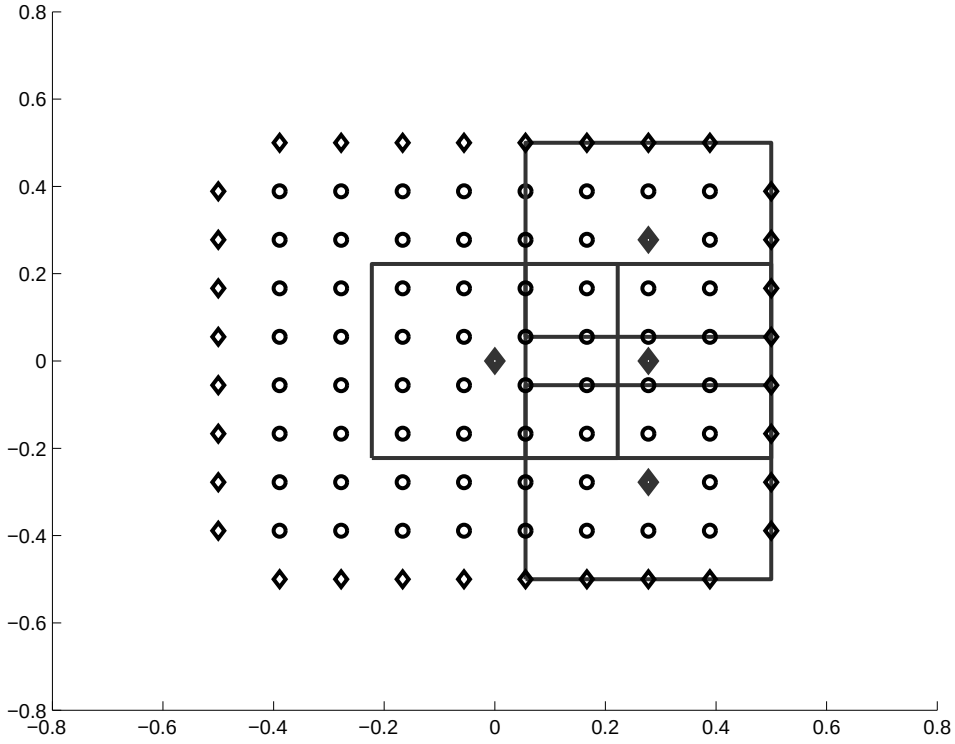

Figure 4.3: Adjacent windows of size L=5 and overlap O=1.5

Note, however, that given some other arbitrary placement of two $L = 5$ windows which overlap by 1.5 (not at the placements given by the example above),

we are not guaranteed that two points will fall under the overlapping sections of the windows.

Finally, we show an example of the case where we need additional windows to cover the boundary. This case arises when the number of windows N_w in equation 4.22 does not cover the boundary. This happens because we are using the floor operation in equation 4.22. We do this because we prefer instead of placing the windows outside the boundary, to place them exactly on the boundary. Notice with this scheme, when these additional windows are needed, the overlap from the points placed on the boundary with the interior placed points is always greater than O, never less than. Here, we use a 16 by 16 gridding of $(\mathbf{x}, \mathbf{y})$, with L=6.5 and O=1.5.

The optimal window placement scheme shown above using $O = 1.5$ will be used in our examples in solving Helmholtz problems, along with using additional windows placed on the boundary when necessary.

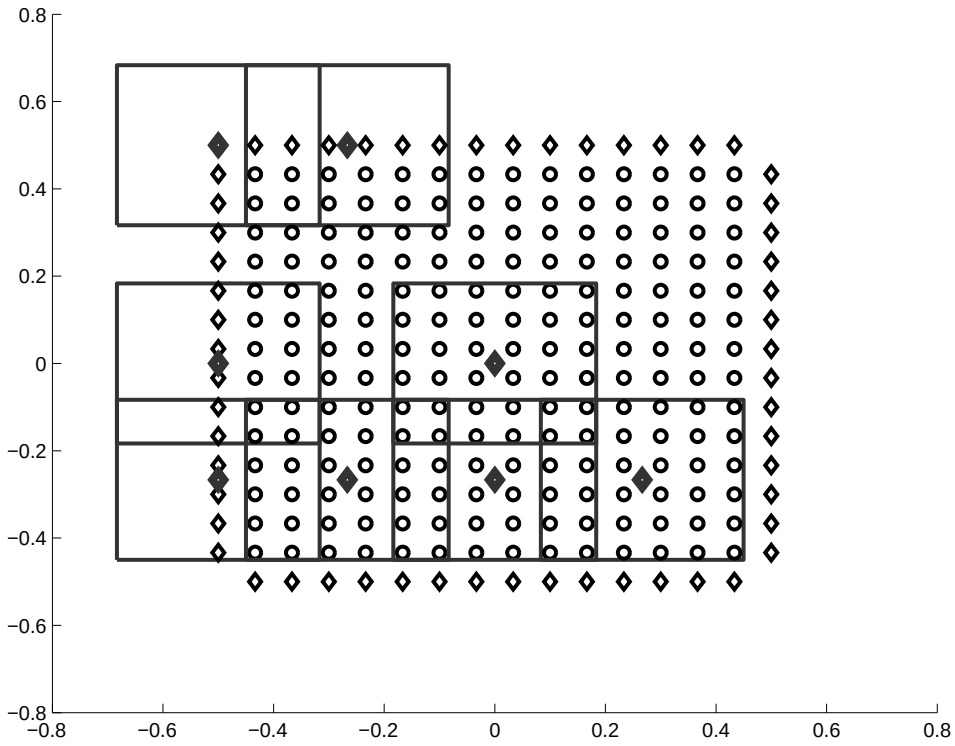

Figure 4.4: A set of windows of size L=6.5 and overlap O=1.5

4.5 Implementation Overview - numerical considerations

Presented in this section are some points to consider when implementing the FMSN method. We will cover a necessary idea to produce working rectangular windows along with some essential steps for sparse memory allocation.

For the rectangular windows to work, we must introduce an additional ϵ on the window width in order to account for floating point arithmetic. In 1d for example, given a window size $L = 5$, say we place a window exactly on a grid point in $\mathbf{x}$. Given that $\mathbf{x}$ is equispaced, then the two end points of the window fall exactly on top of the points that are 2*h away from the center of the window. In floating point arithmetic, these points are not guaranteed to be selected. Thus, we add an epsilon to the window length which guarantees these points fall just inside the window. For point selection, $\epsilon = \frac{h}{1000}$ works well.

The same idea must be applied again when doing the mapping $\phi(x)$ from the points under the window to $[-1, 1]$ to be used in the Chebyshev expansions. The problem is in possibly mapping a point which falls at the end point of the window (which includes the ϵ) to 1 or slightly over 1. We then add instead of just ϵ to the transformation used in mapping, we add a 2ϵ. This guarantees that no points selected by the window will be exactly on 1 in the Chebyshev basis, which could potentially create NaN results.

Finally, a consideration in allocating memory for the grand matrices M_{Gu} and M_{Gf}. We must first determine how many rows each of these matrices has. This can only be known by doing a sweep of all the windows, and seeing how many points fall under each window. We can then total this number up, seen as 'rowcnt' in the attached MATLAB code in the appendix. In addition to just getting the

size of the grand matrices, during this sweep we can collect the necessary index information which will setup the sparse matrix indexing that is required to put the local solutions for M_i into the grand matrices The number of columns in M_{Gf}, M_{Gu}, and M_{Gb}.

4.6 Numerical Results

We present the numerical results for FMSN applied to Elliptic PDEs in this section. Two of the PDEs are Helmholtz problems, and one is a div-curl problem. The results contain the maximum relative error, measured only at the interior discretization nodes, $(\mathbf{x^{int}}, \mathbf{y^{int}})$.

$$error = \frac{\|u(\mathbf{x^{int}}, \mathbf{y^{int}}) - \mathbf{u}^{int}_{solved}\|_\infty}{\|u(\mathbf{x^{int}}, \mathbf{y^{int}})\|_\infty} \tag{4.23}$$

All experiments for FMSN were run using MATLAB, on a machine with 8GB of RAM.

The first problem, which we will call 'good Helmholtz' is given by

$$\nabla^2 u - u = f, \; in \, \Omega \tag{4.24}$$

$$u = g, \; in \, \partial\Omega \tag{4.25}$$

$$u = \frac{1}{1 + x^2 + y^2} \tag{4.26}$$

N	L=3 FD	L=7 FD	L=13 FD	L=17 FD	L=5 FMSN
30	9.5e-4	4.1e-5	1.6e-07	9.3e-09	1.2e-07
100	8.2e-5	8.5e-7	2.8e-10	4.0e-11	6.9e-10
200	2.0e-5	1.1e-7	4.8e-11	7.6e-11	4.5e-11
500	1.2e-5	7.7e-9	3.1e-10	9.8e-10	6.3e-12

Table 4.1: Relative Error for MSNFD vs FMSN, 'good Helmholtz'

on the region $(x, y) \in [-0.5, 0.5], [-0.5, 0.5]$. The region is equisampled with N samples in both directions, along with removing the four corner boundary points. We show results for a window size $L = 5$, compared with the family of results from the Finite Difference MSN method. A window overlap of 1.5 as defined in equation 4.22 is used for window placement. The Sobolev parameter s is set to 15 in all experiments. We can see the method attains a very high accuracy at a low number of samples.

The next problem we consider is the 'hard Helmholtz' problem, given below. We have added strong high frequency term making spectrum two-sided, along with adding Neumann boundary conditions. In addition, the solution u is extremely rough. Note that accuracies would be achieved similar to those of the first problem if the underlying function u was the same. In this case, however, we need at least $N = 100$ points to even begin to approach what could be seen as the Nyquist sampling rate of the underlying function. The FMSN solver at $L = 13$ is on par with the FDMSN solver at $L = 13$.

N	L=3 FD	L=7 FD	L=13 FD	L=17 FD	L=13 FMSN
30	2.8e-01	7.8e-01	9.0e+0	2.5e+02	7.412667e+00
100	1.9e-01	1.3e-01	1.6e-01	2.1e-01	3.530832e+00
200	4.4e-01	1.0e-02	5.2e-03	2.9e-03	2.792513e-03
500	2.2e-02	1.1e-03	3.3e-05	2.6e-06	1.366790e-05

Table 4.2: Relative Error for MSNFD vs FMSN, 'hard Helmholtz'

$$\nabla^2 u + 10000u = f, \; in \; \Omega \tag{4.27}$$

$$\nabla u.\hat{n} = g, \; in \; \partial\Omega \tag{4.28}$$

$$\begin{aligned} u = & \frac{sin(10x + 201y^2)}{1 + 900(x^2 + y - 0.1)^2} \\ & + \frac{1}{1 + 721(x + y - 0.3^2)} \\ & + \frac{e^{-x^2}}{1 + 1000(x + y^2 - 0.25)^2} \\ & + \frac{1}{1 + 1120(x^2 + y^2 - 0.5)^2} \end{aligned} \tag{4.29}$$

We can get a loose handle on the order of the method, based on the window size L. As an initial estimate, the order of the method for the good Helmholtz problem is around 5.5 for lower values of N and is close to 3.5 for larger N, for a $L = 5$ window. For the hard Helmholtz problem, the order is at 4 for larger values of N with a $L = 13$ window.

Finally, we show results a div-curl on the square region $(x, y) \in [-0.5, 0.5], [-0.5, 0.5]$, using a smooth function to show that the method actually works. In 2d, the solu-

tion is now a vector field with two values $(\mathbf{u}_{xg}, \mathbf{u}_{yg})$ at every point on the interior discretization points. We name the components $\mathbf{u}_{xg}$ and $\mathbf{u}_{yg}$, which not to be confused with the partial derivatives of some 2d function $u(x, y)$ which are sometimes written u_x and u_y.

The boundary conditions are setup such that on the top three sides of the square, we have given the normal component only, while on the bottom side, only the tangential component is known. This requirement comes from a constraint on the existence of solutions to the PDE itself. Well-posedness of planar div-curl systems is the topic of [**?**], where many more examples and details can be found. The div-curl PDE we solve numerically is given by:

$$\frac{\partial u_{xg}(x,y)}{\partial x} + \frac{\partial u_{yg}(x,y)}{\partial y} = f_{div}\, in\, \Omega \tag{4.30}$$

$$-\frac{\partial u_{xg}(x,y)}{\partial y} + \frac{\partial u_{yg}(x,y)}{\partial x} = f_{curl}\, in\, \Omega \tag{4.31}$$

$$u_{xg} = g1,\ in\, \partial\Omega_1 \tag{4.32}$$

$$u_{yg} = g2,\ in\, \partial\Omega_2 \tag{4.33}$$

$$u_{xg}(x,y) = \frac{1}{1+x^2+y^2} \tag{4.34}$$

$$u_{yg}(x,y) = x^2 - 2y^2 + xy - x + 1 \tag{4.35}$$

where $\partial\Omega_1 = \partial\Omega \cap \{\{x = -0.5\} \cup \{x = 0.5\} \cup \{y = -0.5\}\}$

and $\partial\Omega_1 = \partial\Omega \cap \{y = 0.5\}$.

N	rel err	cond no	LS residual
7	4.3e-4	1.2e5	5.2e-6
15	1.7e-5	1.6e4	6.7e-8
30	3.5e-8	2.1e5	2.2e-9
60	1.3e-9	3.8e5	2.3e-10

Table 4.3: FMSN div-curl results, smooth solution

We treat the other components on the boundary as if they don't exist, and only solve for (u_{xg}, u_{yg}) on the interior. Also, we have chosen a window width of $L = 5$ and an overlap of $O = 1.5$.

This concludes the preliminary collection of results for FMSN. We have shown that we have a higher order method, which can solve planar div-curl problems. Now it remains to be shown that method will work over a wide class of PDE problems and dimension. In future experiments, more window sizes must be tested, and we must look into the behavior of the condition number for large N. In order to get condition number estimates for such large matrices, we need to move to a supercomputer setting. This will also enable us to obtain results for very large values of N.

Chapter 5

Extensions and Future Work

Many experiments still need to be done using FMSN. These include going through each PDE and detailing the order of convergence, and other results with varying window sizes, overlap, and grid densities.

The real advantage this method has is that it is a least squares method. The method should be tested on div-curl problems of varying geometries and boundary conditions. Papers have been written on the well-posedness of div-curl systems, this will be a source of direction.

We shall also apply FMSN to a whole host of non-elliptic PDEs. This can possibly take the form of time stepping and higher dimensional approaches.

As we become more confident in the method, we can begin applying it to real world applications. At this moment however, much of the above explorations still need to be done.

Bibliography

[1] S. Chandrasekaran, K. R. Jayaraman, M. Gu, H. N. Mhaskar, and J. Moffitt. Msnfd : A higher order finite difference method for solving elliptic pdes on scattered points. *Technical Report*, 2011.

[2] S. Chandrasekaran, K. R. Jayaraman, J. Moffitt, H. N. Mhaskar, and S. Pauli. Minimum sobolev norm schemes and applications in image processing. volume 7535, page 753507. SPIE, 2010.

[3] S. Chandrasekaran and H. N. Mhaskar. A construction of linear bounded interpolatory operators on the torus. *ArXiv e-prints*, Nov. 2010.

[4] P. J. Davis. Interpolation and approximation. 1963.

[5] J. F. Epperson. On the runge example. *Am. Math. Monthly*, 94:329–341, April 1987.

[6] G. Golub and C. Loan. *Matrix computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, 1996.

Appendices

.1 Global Approach - 1d Boundary Value Problem

In this section, we present an ODE solver which uses the same principles as the FMSN PDE solver. We make a separate presentation here to explicitly write out the exact details of the solver using a second order ODE. Here we cover the basic building block of our approach through the example of solving globally a 1d boundary value problem with Dirichlet boundary conditions. In this section, we disregard numerical and memory considerations as a compromise in order to simply demonstrate basic idea of the method. Thus, the method described here is incomplete and is not suitable for use in solving larger problems to high accuracy. One issue is due to memory constraints inherent in forming the dense matrix to be factored for the global solution. The other is due to requirement of an accurate QRV factorization of this matrix. Moving to the local method presented in the following sections remedies both of these issues. The local method uses a generalized version of what is presented in this section in a way which solves numerically an ODE or PDE with minimum Sobolev norm.

The differential equation is denoted by the differential operator $\mathcal{L}$. We will solve the following second order differential equation for u on the interval $[-1, 1]$, with Dirichlet boundary conditions. Thus, $u(-1)$ and $u(1)$ are given.

$$\mathcal{L}(u(x)) = u''(x) - u(x) = f(x), x = [-1, 1] \tag{.1}$$

$$u(-1) = \alpha_1, u(1) = \alpha_2 \tag{.2}$$

The following is notation for the discretization. Assume for simplicity we are discretizing over the region $x = [-1, 1]$ using N equispaced discretization nodes. Each node is an element of the vector $\mathbf{x}$ and is denoted x_k, $k = 1, 2...N$. x_1 and x_N represent the two boundary nodes at $x = -1$ and $x = 1$. The values of the differential equation are stored in the vector $\mathbf{f}$ and are denoted f_k, $k = 2, 3...N - 1$. The values in $\mathbf{f}$ are only known on the interior nodes $x_2...x_{N-1}$. The values of the solution at the nodes $\mathbf{x}$ are stored in the vector $\mathbf{u}$. The elements of $\mathbf{u}$ are denoted u_k, $k = 1, 2...N$. The boundary values of $\mathbf{u}$ are assumed to be known and are denoted u_1 and u_N. The problem is now to solve the following constrained optimization problem:

$$\min_{\{B\mathbf{a}=\mathbf{z}\}} \|D_s\mathbf{a}\|_2^2 \tag{.3}$$

Where $B\mathbf{a} = \mathbf{z}$ represents:

$$\begin{bmatrix} \mathcal{L}V(\mathbf{x}_{int}) \\ V(x_1) \\ V(\mathbf{x}_{int}) \\ V(x_N) \end{bmatrix} \mathbf{a} = \begin{bmatrix} \mathbf{f} \\ u_1 \\ \mathbf{u}_{int} \\ u_N \end{bmatrix} \tag{.4}$$

The Chebyshev vandermonde matrix V is assumed to have sufficiently high order M as defined in equation 3.20, and contains a row for every node in $\mathbf{x}$. $V(x_1)$ and $V(x_N)$ are the rows in V which correspond to the boundary points, while we denote $V(\mathbf{x}_{int})$ to mean the rows corresponding to the interior points $x_2 ... x_{N-1}$. $\mathcal{L}V(\mathbf{x}_{int})$ represents the differential operator applied to the Chebyshev polynomials in the Vandermonder matrix. Thus, in this example, $\mathcal{L}V(\mathbf{x}_{int}) = V''(\mathbf{x}_{int}) - V(\mathbf{x}_{int})$. We write out the second and fourth blocks of B and z separately because they represent the boundary equation. Some other differential operator could be applied to these boundary rows, but for this example we are solving a boundary value problem with Dirichlet conditions.

Assume we have some $\mathbf{u}$ which satisfies the constraint. We rewrite $D_s\mathbf{a}$ in terms of $\mathbf{z}$ and plug this back into the minimization. The minimization is now in terms of $\mathbf{z}$, and thus $\mathbf{u}$. The problem becomes an unconstrained least squares minimization over $\mathbf{u}$.

We begin by inserting the Sobolev weight matrices $D_s^{-1}D_s$ into $B\mathbf{a} = \mathbf{z}$, followed by taking a QRV factorization of BD_s^{-1}. We can assume for now that an SVD is sufficient for small problems. Numerical considerations are addressed in the following sections.

$$BD_s^{-1}D_s\mathbf{a} = \mathbf{z} \tag{.5}$$

$$Q_1R_1W_1 = BD_s^{-1} \tag{.6}$$

Substituting back and taking Q_1 and R_1 to the right hand side:

$$Q_1R_1W_1D_s\mathbf{a} = \mathbf{z} \tag{.7}$$

$$W_1D_s\mathbf{a} = R_1^{-1}Q_1^T\mathbf{z} \tag{.8}$$

We can plug this back into the minimization problem since W_1 is orthogonal, and notice the problem is now over $\mathbf{u}$, $\mathbf{a}$ has been eliminated from the minimization.

$$\min_{\mathbf{a}} \|D_s\mathbf{a}\|_2^2 = \min_{\mathbf{a}} \|W_1D_s\mathbf{a}\|_2^2 \tag{.9}$$

$$= \min_{\mathbf{u}} \|R_1^{-1}Q_1^T\mathbf{z}\|_2^2 \tag{.10}$$

Taking the terms inside the minimization in equation .10, we rename $R_1^{-1}Q_1^T$ as M and partition it into two column blocks M_f and M_u corresponding to the rows in $\mathbf{z}$ containing $\mathbf{f}$ and $\mathbf{u}$, respectively. Note, M is square and has size $(2N-2, 2N-2)$. $N-2$ rows are from $\mathbf{f}$ and N are from $\mathbf{u}$.

$$R_1^{-1}Q_1^T = M = \begin{bmatrix} M_f & M_u \end{bmatrix} \tag{.11}$$

Using the above partition, we can rewrite the minimization problem in equation .10 as:

$$\min_{\mathbf{u}_{int}} \left\| \begin{bmatrix} M_f & M_u \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ u_1 \\ \mathbf{u}_{int} \\ u_N \end{bmatrix} \right\|_2^2 \tag{.12}$$

Recall, u_1 and u_N are the Dirichlet boundary condition values and are known, thus we are minimizing over $\mathbf{u}_{int}$. We can split the matrix multiplication in equation .12 even further[1]. It should also be apparent now that we are minimizing over $\mathbf{u}_{int}$, as the values of $\mathbf{u}$ on the boundary are given.

$$\min_{\mathbf{u}_{int}} \|M_f\mathbf{f} + M_u(*, 2:N-1)\mathbf{u}_{int} + u_0 M_u(*,1) + u_{N-1}M_u(*,N)\|_2^2 \tag{.13}$$

$$\min_{\mathbf{u}_{int}} \|M_u(*, 2:N-1)\mathbf{u}_{int} - (-M_f\mathbf{f} - u_0 M_u(*,1) - u_{N-1}M_u(*,N))\|_2^2 \tag{.14}$$

Written in this form, we can see now that equation .14 is a standard least squares minimization problem, equivalent to the well known problem $Ax = b$ when A is a tall matrix. Note the matrix $M_u(*, 2:N-1)$ is tall and has dimensions (2N-2,N-2). The values for $\mathbf{u}_{int}$ can now be solved for by any of the standard least squares matrix solutions [6], for example by using a QR factorization.

.2 Numerical Considerations

In the previous section we used a global approach to finding the solution which has minimum Sobolev norm of an ODE. The problem with this approach can be seen when we increase N. Here, the dense matrix M will grow with N. For large problems in 2d, the QRV factorization not only becomes too expensive, but also highly inaccurate. To see why, consider the matrix undergoing the factorization, BD_s^{-1}. In D_s^{-1}, the high frequency terms become exponentially small, thus rendering the matrix BD_s^{-1} to be highly ill conditioned for larger values of s. Specialized techniques have been developed to deal with this situation, as can be found in [1],

[1]The notation $(*, 1:3)$ following a matrix is used to denote selecting the first 3 columns of that matrix, for example.

see the sections on CODA, the Complete Orthogonal Decomposition Algorithm. As a simpler work around, we stick to using an SVD as the QRV factorization and propose using the local approach presented in the following section as a way to keep the problem at bay.

.3 Local Approach - 1d Boundary Value Problem

We now solve the same ODE that was used in the global approach. This time instead of using the whole grid of discretization nodes $\mathbf{x}$ to form the M matrix we place a new grid down which we call the specification points. The k specification points will be stored in the vector $\mathbf{c}$. A specification point c_l will be the center point for which we place a window of some width which will grab the discretization nodes $\mathbf{x}$. We will still be solving for the ODE at the points in $\mathbf{x}$. Locations in $\mathbf{c}$ can match locations in $\mathbf{x}$, including being on the boundary.

One way to specify the size of the local window is in relation to the grid density of $\mathbf{x}$, given $\mathbf{x}$ is an equispaced grid. Let h be the distance between two samples in $\mathbf{x}$. Then we can define a window width of L samples for example. Let C be the set of values in $\mathbf{x}$ that fall in this window placed at some location stored in $\mathbf{c}$.

$$C_l = \{x_n : \|x_n - c_l\|_\infty < \frac{h(L-1)}{2}\}, l = 1, 2...k \tag{.15}$$

The idea now is to place windows such that the whole region of points in $\mathbf{x}$ is covered. We are free to choose the window sizes and locations, we will see this effects the accuracy and solve time in the discussion to follow. For now, we show how to setup the large system which will solve the ODE with minimum Sobolev norm.

Each window placed at c_l will select a set of discretization points from $\mathbf{x}$. Some windows may fall over only interior points, other will select both boundary and interior points, depending on their location c_l. Knowing the corresponding equations for the interior and boundary at every point in $\mathbf{x}$, we can setup a local solve, treating it identically to how we setup the global solution in the previous section. One extra step needs to be added, and that is the affine transformation which maps points from the windowed neighborhood set C_l to $[-1, 1]$.

Thus, each window will have its own set of local equations given by the region in which it is placed, and thus its own Sobolev norm to minimize. We can now pose this as the following global optimization problem:

$$\min_{\{\mathbf{u}|B_i\mathbf{a}_i=\mathbf{z}_i\}} \sum_{i=1}^{k} \|D_s\mathbf{a_i}\|_2^2 \tag{.16}$$

Where $B_i\mathbf{a}_i = \mathbf{z}_i$ represents the local set of equations given by the points that fall under the k windows centered at c_i, $i = 1, 2...k$. Thus we are asking that the sum of the Sobolev norms given locally from each $\mathbf{u}_i$ be minimum.

Proceeding by example, if the first window c_1 contains only the left boundary and some close by interior points, then $B_1\mathbf{a}_1 = \mathbf{z}_1$ would mean:

$$\begin{bmatrix} \mathcal{L}V(\phi_1(\mathbf{x}_{int1})) \\ V(\phi_1(x_1)) \\ V(\phi_1(\mathbf{x}_{int1})) \end{bmatrix} \mathbf{a} = \begin{bmatrix} \mathbf{f} \\ u_1 \\ \mathbf{u}_{int1} \end{bmatrix} \tag{.17}$$

Where $\mathbf{x}_{int1}$ are the interior points which fall under window 1, which is centered at c_1. Note, c_1 is allowed to be a boundary point. We can not simply use the points in $\mathbf{x}$ which fall under the window in the Vandermonde matrix. For example, if we were solving on a region outside of $[-1, 1]$, these points are not even defined for the Chebyshev polynomials. We must use an affine transformation to map the points under the window into $[-1, 1]$. To form a consistent mapping, we map the leftmost reaching value of the window to -1 and the rightmost reaching value to 1, not the left and right most point under the window. For now, we assume a constant window size. Future work could include adaptive algorithms where window sizes vary.

We now proceed identically to the idea behind the global method. We form M_{u1} and M_{f_1}, using 1 to denote from the first window. However, instead of solving for $\mathbf{u}$ locally here, we stop and save the matrices M_{u1} and M_{f_1} and place them into the larger sparse grand matrices M_{Gu} and M_{Gf}. We repeat the process for all k windows. Each new window gets its own set of rows in M_{Gu} and M_{Gf}, but the columns of both M_{Gu} and M_{Gf} are indexed in relation to the points in $\mathbf{x}$. According to how much the windows overlap, columns will be shared. A banded structure arises for 1d problems in both grand matrices M_{Gu} and M_{Gf} assuming that our window has slid from the left boundary to the right boundary. In higher dimensions, the sparsity pattern will not be as simple.

$$\min_{\mathbf{u}} \left\| \begin{bmatrix} M_{Gf} & M_{Gu} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \mathbf{u} \end{bmatrix} \right\|_2^2 \tag{.18}$$

Where M_{Gu} and M_{Gf} are the following sparse matrices.

$$M_{Gu} = \overbrace{\begin{bmatrix} [M_{u1}] & & & \\ & [M_{u2}] & & \\ & & [M_{u3}] & \\ & & & \ddots \\ & & & [M_{uk}] \end{bmatrix}}^{\text{columns indexed as all points in } \mathbf{x}}, M_{Gf} = \overbrace{\begin{bmatrix} [M_{f_1}] & & & \\ & [M_{f_2}] & & \\ & & [M_{f_3}] & \\ & & & \ddots \\ & & & [M_{f_k}] \end{bmatrix}}^{\text{columns indexed as the interior points in } \mathbf{x}} \tag{.19}$$

We again find the resulting solution has been reduced to a least squares problem. The first and last columns of M_{Gu} multiplied by the boundary conditions u_1 and u_N, respectively, should be taken to the right hand side. Then $\mathbf{u_{int}}$ can be solved for using any standard matrix algebra technique, for example using by a QR factorization of the remaining columns of the matrix M_{Gu}. This is assuming all the values of $\mathbf{f}$ are known, and are thus also multiplied with M_{Gf} and taken to the right hand side.

Using the local approach gives us control over the sparsity and size of the matrix which will must invert, M_{Gu}. This also controls the amount of computation time required in assembling the matrix. Using a smaller window size will reduce the number of non-zero terms, along with the number of rows. Using fewer specification points for the windows will reduce the number of rows. The number of columns is fixed as the number of points in $\mathbf{x}$.

In choosing windows sizes and locations, it is important that no points in $\mathbf{x}$ are not covered by a window, or we will end up with a zero column in M_{Gu}, given that we setup M_{Gu} to have a column for every point in $\mathbf{x}$ before adding the local computations. We also at this point make the conjecture that for second order problems, a minimum window overlap of two points is required. Likewise for fourth order problems, we would need four, though this has not yet been proven mathematically.

.4 MATLAB code

```
% FMSN PDE Solver
% 9/7/11
% Joseph Moffitt

clear
%clc

%u = @(x,y) ((sin(10*x+201*y.^2)./(1+900*(x.^2+y-0.1).^2)) + (1./(1+...
```

```
%721*(x+y-0.3).^2)) + (exp(-(x.^2))./(1+1000*(x+y.^2-0.25).^2...
%)) + (1./(1+1120*(x.^2+y.^2-0.5).^2)));
%u = @(x,y) 1/(1+100*x^2+100*y^2);
u = @(x,y) 1./(1+x.^2+y.^2)


usym = sym(u);
ux = diff(usym,'x');
uxx = diff(ux,'x');
uy = diff(usym,'y');
uyy = diff(uy,'y');
%make sure the PDE in line 251 matches the following
f = uxx+uyy-usym;
%f = uxx+uyy+10000*usym;

shape = 'Square_geom';
Ntest = [10];
stest = [15];
mkeeptest = [5];

errall = zeros(length(Ntest),length(stest),length(mkeeptest));
condall = zeros(length(Ntest),length(stest),length(mkeeptest));

iN = 1;
iS = 1;
iM = 1;

tic
for N = Ntest
for s = stest


Nx = N;
Ny = N;

fprintf('OK READY\n')
fprintf('N = %d\n',N)

width = 1.0;
height = 1.0;
h_w = width/(Nx-1);
h_h = height/(Ny-1);
```

```
epsl_w = h_w/1000;
epsl_h = h_h/1000;
L=5;
OVL = 1.5;
winl = (L-1)*h_w;
WinSPACE = winl-OVL*h_w;
numwins = 2*floor(width/(2*WinSPACE))+1;

ninterior = N^2;
nboundary = 4*N;

%yi first to mimick the python code!!!!!!!
[yi xi xb yb h] = generate_grid(ninterior, nboundary, shape);
%Now, call the FMSN PDE Solver, with the given boundary.
% The following lines makes this code mimick the python code
xb = fliplr(xb);
yb = fliplr(yb);
remcorners = [1, Nx,(Nx+Ny-1),(2*Nx+Ny-2)];
xb(remcorners) = [];
yb(remcorners) = [];

%xb = xb*width;
%yb = yb*height;
%xi = xi*width;
%yi = yi*height;

Ni = length(xi);
Nb = length(xb);
Ntot = Ni+Nb;

xb = xb';
yb = yb';

x = [xi;xb];
y = [yi;yb];

Nside = Nb/4;
% !!!!!!!!!! SETUP BOUNDARY CONDITIONS HERE !!!!!!!!!!!!
%  Make sure the BC matches the one in line 263
%BC_V = [zeros(Nside,1) ones(Nside,1); ones(Nside,1) zeros(Nside,1...
%); zeros(Nside,1) ones(Nside,1); ones(Nside,1) zeros(Nside,1)];
```

```
if ((numwins-1)*WinSPACE/2+winl/2)<width/2
    add2 = 2;
else
    add2 = 0;
end

if add2 == 0
NxW = numwins;
NyW = numwins;

nWinterior = (NxW)^2;
nWboundary = 4*(NxW);

%yi first to mimick the python code
[yi_W xi_W xb_W yb_W h_W]=generate_grid(nWinterior, nWboundary, shape);
yi_W=(yi_W*2)*((numwins-1)/2)*WinSPACE;
xi_W=(xi_W*2)*((numwins-1)/2)*WinSPACE;
xb_W=(xb_W*2)*((numwins-1)/2)*WinSPACE;
yb_W=(yb_W*2)*((numwins-1)/2)*WinSPACE;
end
if add2 == 2
NxW = numwins;
NyW = numwins;
nWinterior = (NxW)^2;
nWboundary = 4*(NxW);
%yi first to mimick the python code
[yi_W xi_W xb_W yb_W h_W]=generate_grid(nWinterior, nWboundary, shape);
yi_W=(yi_W*2)*((numwins-1)/2)*WinSPACE;
xi_W=(xi_W*2)*((numwins-1)/2)*WinSPACE;
xb_W=(xb_W*2)*((numwins-1)/2)*WinSPACE;
yb_W=(yb_W*2)*((numwins-1)/2)*WinSPACE;
xb_W = fliplr(xb_W);
yb_W = fliplr(yb_W);
yi_W = [yi_W; yb_W'];
xi_W = [xi_W; xb_W'];
xb_W = [-0.5 xb_W(1:(NxW)) 0.5*ones(1,NyW+2) fliplr(xb_W(1:(NxW...
    ))) -0.5*ones(1,NyW+1)];
yb_W = [-0.5*ones(1,NyW+2) yb_W((NxW):(NxW+NyW-1)) 0.5*ones(1,NyW...
    +2) fliplr(yb_W((NxW):(NxW+NyW-1)))];

end
```

```
Ni_W = length(xi_W);
Nb_W = length(xb_W);
NtotW = Ni_W+Nb_W;
xb_W = xb_W';
yb_W = yb_W';
xW = [xi_W;xb_W];
yW = [yi_W;yb_W];

if L == 3
 Ni_W = length(xi);
 Nb_W = length(xb);
 NtotW = Ni_W+Nb_W;
 xb_W = xb;
 yb_W = yb;
 xW = [xi;xb];
 yW = [yi;yb];
end

%the following plots our window placement setup
%{
figure
hold on;
scatter3(xi, yi, ones(length(xi),1),'ko','LineWidth',2);
plot3(xb, yb, ones(length(xb),1), 'kd','LineWidth',2);
for i = 1:length(xW)
plot3(xW(i), yW(i), 1,'d','Color',[0.2 0.2 0.2],'LineWidth',4);
plot([xW(i)-h_w*(L-1)/2,xW(i)-h_w*(L-1)/2,xW(i)+h_w*(L-1)/2,...
xW(i)+h_w*(L-1)/2,xW(i)-h_w*(L-1)/2], [yW(i)-h_h*(L-1)/2,...
yW(i)+h_h*(L-1)/2,yW(i)+h_h*(L-1)/2,yW(i)-h_h*(L-1)/2,...
yW(i)-h_h*(L-1)/2],'Color',[(i+length(xW)/2)/(length(xW)*2) (i...
+length(xW)/2)/(length(xW)*2) (i+length(xW)/2)/(length(xW)*2)])
end
axis([-0.8 0.8 -0.8 0.8])
hold off;
%}


%Find the windows and indices.
%Store in a list
list_ind = [];
li_u = [];
li_g = [];
```

```
li_f = [];
rowcnt = 0;
tic
for k= 1:NtotW
   Lwb_x = xW(k)-h_w*(L-1)/2-epsl_w;
   Rwb_x = xW(k)+h_w*(L-1)/2+epsl_w;
   Lwb_y = yW(k)-h_h*(L-1)/2-epsl_h;
   Rwb_y = yW(k)+h_h*(L-1)/2+epsl_h;
   xy_w_ind=find(x >= Lwb_x & x <= Rwb_x & y >= Lwb_y & y <= Rwb_y);
   interior_cnt = nnz(xy_w_ind<=Ni);
   boundary_cnt = nnz(xy_w_ind>Ni);
   % the following lists will GROW inside the loop
   list_ind = [list_ind; length(xy_w_ind); xy_w_ind];
   li_f = [li_f interior_cnt];
   li_u = [li_u interior_cnt];
   li_g = [li_g boundary_cnt];

   rows_LV_V_BV = interior_cnt+interior_cnt+boundary_cnt;
   rowcnt = rowcnt+rows_LV_V_BV;
end
gatherinfotime = toc;

li_sum = li_f+li_u+li_g; %each entry is the number of rows of each Mi
f_cnt_win = li_f.*li_sum;
u_cnt_win = li_u.*li_sum;
g_cnt_win = li_g.*li_sum;

iMf = zeros(sum(f_cnt_win),1); %to be used for sparse indexing
jMf = zeros(sum(f_cnt_win),1); %to be used for sparse indexing
Mf_data = zeros(sum(f_cnt_win),1); %data to be filled into the Mf
iMu = zeros(sum(u_cnt_win),1);
jMu = zeros(sum(u_cnt_win),1);
Mu_data = zeros(sum(u_cnt_win),1);
iMg = zeros(sum(g_cnt_win),1);
jMg = zeros(sum(g_cnt_win),1);
Mg_data = zeros(sum(g_cnt_win),1);

Mcheck = [];
ind_find = 1;
rowstop = 0;
ind_umat = 0;
ind_fmat = 0;
```

```
ind_gmat = 0;

tic
for k = 1:NtotW
   fprintf('-indx %d/%d-\n',k,NtotW)
   num_points = list_ind(ind_find);
   xy_w_ind = list_ind(ind_find+1:ind_find+1+num_points-1);

   ind_interior = find(xy_w_ind<=Ni);
   ind_boundary = find(xy_w_ind>Ni);

   x_w = x(xy_w_ind);
   y_w = y(xy_w_ind);

   Lwb_x = xW(k)-h_w*(L-1)/2-2*epsl_w;
   Rwb_x = xW(k)+h_w*(L-1)/2+2*epsl_w;
   Lwb_y = yW(k)-h_h*(L-1)/2-2*epsl_h;
   Rwb_y = yW(k)+h_h*(L-1)/2+2*epsl_h;

   phi_x=2/(Rwb_x-Lwb_x);
   phi_y=2/(Rwb_y-Lwb_y);

   x_w_phi = phi_x*(x_w-Lwb_x)-1;
   y_w_phi = phi_y*(y_w-Lwb_y)-1;


   maxgridsize = 1e6;
   tol = 1/maxgridsize;
   dxc = bsxfun(@minus, acos(x_w_phi), acos(x_w_phi)');
   dyc = bsxfun(@minus, acos(y_w_phi), acos(y_w_phi)');
   dx = sqrt(dxc.^2+dyc.^2);
   M = round(3*pi/min(min(dx(dx>tol))));
   clear dxc dyc dx;
   Mx = M;
   My = M;
   %Mcheck = [Mcheck M];
   D_inv = zeros(Mx*My,1);
   for l=0:Mx-1
     D_inv((l*My + 1):(l+1)*My,1) = (1+l^2+(0:My-1).^2).^(-s/2);
   end

   Vxx = zeros(num_points, Mx*My);
```

```
Vx = zeros(num_points, Mx*My);
Vyy = zeros(num_points, Mx*My);
Vy = zeros(num_points, Mx*My);
V = zeros(num_points, Mx*My);
for t=1:num_points
  Vxx(t,:) = kron(([0:Mx-1].^2.*cos([0:Mx-1]*acos(x_w_phi(t))))/...
      (x_w_phi(t)^2 - 1) + ([0:Mx-1]*x_w_phi(t).*sin([0:Mx-1]*...
      acos(x_w_phi(t))))/(1 - x_w_phi(t)^2)^(3/2),...
      cos(acos(y_w_phi(t))*[0:My-1]));
  Vx(t,:) = kron( ([0:Mx-1].*sin([0:Mx-1]*acos(x_w_phi(t))))/...
      (1 - x_w_phi(t)^2)^(1/2)  , cos(acos(y_w_phi(t))*[0:My-1]));
  Vyy(t,:) = kron(cos(acos(x_w_phi(t))*[0:Mx-1])  ,...
      ([0:My-1].^2.*cos([0:My-1]*acos(y_w_phi(t))))/...
      (y_w_phi(t)^2 - 1) + ([0:My-1]*y_w_phi(t).*sin([0:My-1]*...
      acos(y_w_phi(t))))/(1 - y_w_phi(t)^2)^(3/2));
  Vy(t,:) = kron(cos(acos(x_w_phi(t))*[0:Mx-1]), ([0:My-1].*...
      sin([0:My-1]*acos(y_w_phi(t))))/(1 - y_w_phi(t)^2)^(1/2));
  V(t,:) = kron(cos(acos(x_w_phi(t))*[0:Mx-1]), ...
      cos(acos(y_w_phi(t))*[0:My-1]));
end

% !!!!!!! SETUP THE PDE HERE !!!!!!!!
LV = phi_x^2*Vxx(ind_interior,:)+...
    phi_y^2*Vyy(ind_interior,:)-V(ind_interior,:);
%LV = phi_x^2*Vxx(ind_interior,:)+...
phi_y^2*Vyy(ind_interior,:)+10000*V(ind_interior,:);

BV = zeros(length(ind_boundary),Mx*My);
for ibo = 1:length(ind_boundary)

   % !!!!!!! SETUP THE BOUNDARY CONDITIONS HERE !!!!!!!!
   % Use the following line to have Dirichlet boundary conditions:
   BV(ibo,:) = V(ind_boundary(ibo),:);
   % Use the following line to have Neumann boundary conditions:
   %BV(ibo,:) = phi_x*Vx(ind_boundary(ibo),:)*...
   %BC_V((xy_w_ind(ind_boundary(ibo))-Ni),1)+phi_y*...
   %Vy(ind_boundary(ibo),:)*BC_V((xy_w_ind(ind_boundary(ibo))-Ni),2);
end
Vint = V(ind_interior,:);

[dum,ps] = sort(D_inv, 'descend'); %apply a permutation to the rows
clear dum
```

```
Ddiag = diag(D_inv);
A=Ddiag(ps,:)*[LV' Vint' BV'];
[dum,sig1,w1] = svd(A,'econ');
M1 = diag(1./diag(sig1))*w1';

if sum(sum(isnan(M1)),2)>0
    k
    M1
    fprintf('BREAK! ASSEMBLY CREATED a NaN!')
    break
end

indx_rows_M = 1:li_sum(k);
rep_i_f = repmat(indx_rows_M,1,li_f(k));
rep_i_u = repmat(indx_rows_M,1,li_u(k));
rep_i_g = repmat(indx_rows_M,1,li_g(k));

indx_cols_Mf = reshape(repmat(xy_w_ind(ind_interior)',li_sum(k),1),...
    length(ind_interior)*li_sum(k),1);
indx_cols_Mu = reshape(repmat(xy_w_ind(ind_interior)',li_sum(k),1),...
    length(ind_interior)*li_sum(k),1);
indx_cols_Mg = reshape(repmat(xy_w_ind(ind_boundary)',li_sum(k),1),...
    length(ind_boundary)*li_sum(k),1);

Mfd = reshape(M1(:,1:length(ind_interior)),...
    length(ind_interior)*li_sum(k),1);
Mud = reshape(M1(:,length(ind_interior)+1:2*length(ind_interior)),...
    length(ind_interior)*li_sum(k),1);
if li_g(k) ~= 0 % skip when we cover only interior points
Mgd = reshape(M1(:,2*length(ind_interior)+1:li_sum(k)),...
    length(ind_boundary)*li_sum(k),1);
end

  iMf((ind_fmat+1):(ind_fmat+f_cnt_win(k))) = rep_i_f+rowstop;
  jMf((ind_fmat+1):(ind_fmat+f_cnt_win(k))) = indx_cols_Mf;
  Mf_data((ind_fmat+1):(ind_fmat+f_cnt_win(k))) = Mfd;

  iMu((ind_umat+1):(ind_umat+u_cnt_win(k))) = rep_i_u+rowstop;
  jMu((ind_umat+1):(ind_umat+u_cnt_win(k))) = indx_cols_Mu;
  Mu_data((ind_umat+1):(ind_umat+u_cnt_win(k))) = Mud;

if li_g(k) ~= 0 % skip when we cover only interior points
```

```
     iMg((ind_gmat+1):(ind_gmat+g_cnt_win(k))) = rep_i_g+rowstop;
     jMg((ind_gmat+1):(ind_gmat+g_cnt_win(k))) = indx_cols_Mg-Ni;
     Mg_data((ind_gmat+1):(ind_gmat+g_cnt_win(k))) = Mgd;
   end

   rowstop = rowstop+li_sum(k);
   ind_find = ind_find+num_points+1;
   ind_fmat = ind_fmat+f_cnt_win(k);
   ind_umat = ind_umat+u_cnt_win(k);
   ind_gmat = ind_gmat+g_cnt_win(k);
end
assembletime = toc;



ff = matlabFunction(f);
fRHS = ff(xi,yi);
%{
uref = zeros(length(x),1);
for ik = 1:length(x)
uref(ik) = u(x(ik),y(ik));
end
%}


% !!!!!!! SETUP BOUNDARY CONDITIONS HERE !!!!!!!!!
%uxF = matlabFunction(ux);
%uyF = matlabFunction(uy);
%u_BC_funct = [uxF(xb,yb) uyF(xb,yb)];
%u_BC = sum((u_BC_funct.*BC_V),2);
% Use the following for Dirichlet Boundary conditions
u_BC = u(xb,yb);

tic

Mg = sparse(iMg,jMg,Mg_data,rowcnt,Nb,rowcnt*L^2);
clear iMg jMg Mg_data
Mgu_BC=Mg*u_BC;
clear Mg u_BC

Mf = sparse(iMf,jMf,Mf_data,rowcnt,Ni,rowcnt*L^2);
clear iMf jMf Mf_data
```

```
MffRHS=Mf*fRHS;
clear Mf fRHS

MM_fg=-MffRHS-Mgu_BC;
clear MffRHS Mgu_BC
formmatrixtime = toc;

Mu = sparse(iMu,jMu,Mu_data,rowcnt,Ni,rowcnt*L^2);
clear iMu jMu Mu_data

tic
uSOL = Mu\MM_fg;
solvtime = toc;
uref = u(xi,yi);
err = max(abs(uSOL-uref))/max(uref);


LSresid = max(abs(((Mu*uref)-(MM_fg))))/max(abs((MM_fg)));
LSresidsolv = max(abs(((Mu*uSOL)-(MM_fg))))/max(abs((MM_fg)));

n_non_z = nnz(Mu);
mnew = sign(abs(Mu));
mutnnz = sum(mnew,2);
maxBW = max(mutnnz);
avgBW = sum(mutnnz)/length(mutnnz);
maxBWrows = sum(maxBW == mutnnz);

tic
RS = qr(Mu,0);
xrand = rand(size(RS,1),20);
xrandcol = max(abs(xrand));
xr1 = xrand./repmat(xrandcol,size(RS,1),1);
bo=RS*xrand;
normRs = max(max(abs(bo)));
xsol = RS\xr1;
normRsi = max(max(abs(xsol)));
wallis = sqrt(pi*(size(RS,1)-1)/2);
condno = wallis*normRs*normRsi;
condnotime = toc;

fprintf('Information about Mu:\n')
fprintf('Number of Non-zeros vs total size of Mu: %d, %d\n',n_non_z,...
```

```
    size(Mu,1)*size(Mu,2))
fprintf('Max bandwidth of Mu = %d, found in %d rows\n',full(maxBW),...
    full(maxBWrows))
fprintf('Avg bandwidth of Mu = %d\n',full(avgBW))
fprintf('Size of Mu = %d x %d \n',size(Mu,1),size(Mu,2))
fprintf('Cond number of Mu = %d\n',condno)
fprintf('LS residual w/ refernce = %d\n',LSresid)
fprintf('LS residual w/ solved soln = %d\n',LSresid)
fprintf('Max absolute relative error =  %d\n',err)
fprintf('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n',err)
fprintf('Information about Timings:\n')
fprintf('Cond no est time =  %d\n',condnotime)
fprintf('Solve time =  %d\n',solvtime)
fprintf('Assemble time =  %d\n',assembletime)
fprintf('Form sparse matrix time =  %d\n',formmatrixtime)
fprintf('Gather indeces time =  %d\n',gatherinfotime)

figure
plot3(xi,yi,(uSOL-uref))

errall(iN,iS,iM) = err;
condall(iN,iS,iM) = condno;

iS = iS+1;
end

iS=1;
iN=iN+1;
end
tottime=toc;
save('err_N_s_m.mat','errall')
save('cond_N_s_m.mat','condall')
save('time.mat','tottime')
```