Domain Partitioning to Bound Moments of Differential Equations Using Semidefinite Optimization

by

Sandeep Sethuraman

Submitted to the School of Engineering in partial fulfillment of the requirements for the degree of Master of Science in Computation for Design and Optimization at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

© Massachusetts Institute of Technology 2006. All rights reserved.
Domain Partitioning to Bound Moments of Differential Equations Using Semidefinite Optimization

by

Sandeep Sethuraman

Submitted to the School of Engineering
on August 10, 2006, in partial fulfillment of the requirements for the degree of Master of Science in Computation for Design and Optimization

Abstract

In this thesis, we present a modification of an existing methodology to obtain a hierarchy of lower and upper bounds on moments of solutions of linear differential equations. The motivation for change is to obtain tighter bounds by solving smaller semidefinite problems. The modification we propose involves partitioning the domain and normalizing each partition to ensure numerical stability. Using the adjoint operator, linear constraints involving the boundary conditions and moments of the solution are developed for each partition. Semidefinite constraints are imposed on the moments, and an optimization problem is solved to obtain the bounds. We have demonstrated the algorithm by calculating bounds on moments of various one-dimensional case differential equations including the Bessel ODE, and Legendre polynomials. In the two-dimensional case we have demonstrated the algorithm by calculating bounds on various PDEs including the Helmholtz equation, and heat equation. In both cases, the results were encouraging with tighter bounds on moments being obtained by solving smaller problems with domain partitioning.

Thesis Supervisor: Pablo A. Parrilo
Title: Associate Professor
Acknowledgments

I would like to express my thanks to a number of individuals for their contributions to this thesis in particular, and my time at MIT as a student.

First, I would like to thank my adviser, Pablo Parrilo, for the many hours of guidance and encouragement. It has been an amazing experience working under his supervision, and I will be forever thankful for this opportunity.

My gratitude extends to Constantine Caramanis, who helped tremendously in understanding his work, and provided thoughtful insights during the course of the project. Professor Bertismas’s encouragement during the course of the project was very useful. I want to thank J. Lofberg for writing and making freely available his excellent software YALMIP [11], and for his guidance and advice during code development. I am grateful to Singapore-MIT Alliance for their financial support.

I would also like to thank my friends Anu, Harish, Ram and Ruchi, with whom I have had some wonderful experiences, and whose friendship I will value for life; Sukant and Ashish, friends who with their good humor made my time at MIT totally unique by making sure I was not stuck in my lab all day and all night. I am grateful to Mabel for helping me with my thesis tremendously, and for proofreading it more times than I myself might have done. Hu Sha, Sun Jiali, Homan, Shawn and Georgios with their friendship and conversation made the days and many nights in LIDS much more enjoyable.

I owe my deepest thanks to my parents Beena and Sethuraman, and my ever loyal sister Lakshmi, for their unconditional support, and their belief in me all these years. They have done the best job of being family when family was needed. I owe them more than I would be able to express.
## Contents

1 Introduction ........................................... 15
   1.1 Standard solution techniques for PDEs .......... 16
       1.1.1 Finite difference method ................. 17
       1.1.2 Spectral methods ...................... 18
       1.1.3 Finite element method ................. 20
   1.2 Moment calculation using semidefinite optimization .... 21

2 Description of the method .................................. 23
   2.1 Basic approach - theory ....................... 23
   2.2 Adjoint operator ............................. 24
   2.3 Definitions .................................. 26
   2.4 Moments and moment matrices ................. 26
   2.5 Semidefinite constraints .................... 28
   2.6 Motivation for modification ................. 31
       2.6.1 Proposed modification ............... 32

3 One dimensional case - ordinary differential equations(ODEs) .......... 33
   3.1 Partitioning and change of domain / domain mapping ........ 34
       3.1.1 Scaling of constant coefficients ........ 36
       3.1.2 Scaling of non-constant coefficients .... 38
       3.1.3 New boundary variables ............... 39
   3.2 Adjoint operator ............................. 40
   3.3 Coupling conditions .......................... 41
5.6.1 Moment matrices .................................................. 77
5.6.2 Boundary moments ............................................... 77
5.7 Recovering the moments .......................................... 78
5.8 Implementation ...................................................... 78

6 Numerical validation ................................................ 79
6.1 Two-dimensional PDEs .............................................. 80
  6.1.1 Poisson's equation .......................................... 80
  6.1.2 Helmholtz equation ......................................... 81
  6.1.3 Non-constant coefficient example .......................... 82
  6.1.4 Heat equation ................................................. 82
6.2 Discussion ........................................................... 83
  6.2.1 Accuracy for lower degree test functions .............. 83
  6.2.2 Numerical problems ......................................... 84
  6.2.3 Reduction of feasible region ................................ 85
  6.2.4 Relaxation of constraints ................................... 86
6.3 Summary ............................................................. 86

7 Conclusions and future directions ................................ 89

A Expansion of terms in the two-dimensional equations ........ 91
List of Figures

3-1 Partitioning the domain into $D$ partitions. Each partition is mapped on to a new domain $[-1, 1]$. The start and end of each partition are indicated by $a_i$ and $b_i$ respectively. ........................................... 35

3-2 Partitioning of the domain into two regions. The dotted line indicates the new boundary with the coupling conditions ................................. 37

3-3 Brief Summary of Algorithm - The original domain is partitioned and each partition is transformed to a normalized scale. Using the adjoint operator linear constraints are generated. Coupling conditions between partitions, and SDP constraints are also written and the optimization problem is solved. Finally, the original moments of the system are recovered by combining the moments corresponding to each partition. 47

4-1 Plot and error in the solution to ODE $u'' + 12u' + 50u = 4, u'(0) = 5, u'(1) = 0$ ......................................................... 54

4-2 Plot and error in the solution to ODE $u'' + 3u' - 4u = 0, u'(0) = -1, u'(1) = 0$ ................................................................. 55

4-3 Plot and error in the solution to ODE $u'' + 10u' + 50u = 24, u'(0) = 5, u'(1) = 0.5$ ............................................................... 56

4-4 Plot and error in the solution to ODE $u'' + 2u' + 401u = 401, u'(0) = -1, u'(1) = -6.872$ ..................................................... 57

4-5 Plot and error in the solution to a 5th order Legendre Polynomials. This example demonstrates that ODEs with polynomial solution do not benefit by partitioning the domain. ............................... 58
4-6 Plot and error in the solution to ODE $x^2u'' + u' + (9x^2 - 1)u = 0$, $u'(0) = 1.5$, $u'(1) = J_0(3) - J_2(3)$. Solution is the Bessel function $J_1(3x)$.

4-7 Gap between upper and lower bound for $8m_0 + 3m_1$ for the ODE $(x^2 - 4)u'' + 3u' + u = 0$

4-8 Time taken to compute the lower and upper bounds on the objective for the ODE: $u'' + 2u' + 401u = 401$

4-9 Graph showing the solution, and the bounds on $m_0$ for the ODE $u_xx + 3u_x - 4u = 0$, $u_x(0) = -1$, $u_x(1) = 0$ with no partitions. The average value does not indicate very well what is happening in the domain.

4-10 Graph showing the solution, and the bounds on $m_0$ for the ODE $u_xx + 3u_x - 4u = 0$, $u_x(0) = -1$, $u_x(1) = 0$ with 4 partitions. The average values of the four partitions indicate better what is happening in the domain.

5-1 Sketch of the partition of the two-dimensional region. The arrows on the dotted line indicate the coupling conditions between partitions.

6-1 Gap between lower bound and upper bound for $m_{1,0}$ of the differential equation $\Delta u = 4 \sinh(x) \cosh(x) + 4 \cosh(y) \sinh(y)$

6-2 Gap between lower bound and upper bound for $m_{0,0}$ of the differential equation $\Delta u + u = 3e^{x+y}$

6-3 Gap between lower and upper bound in value of $m_{1,0}$ of the equation $u_y = x^4u_{xx} + u$

6-4 Gap between lower and upper bound for value of $m_{1,0}$ for the heat equation $u_{xx} - u_{yy} + u = 0$. For the bigger partitions, SeDuMi runs into numerical problems, and stops solving the optimization with a bigger gap between the dual and primal problems.
List of Tables

4.1 Lower and Upper bounds on the value of $8m_0 + 3m_1$ of the solution of ODE $u'' + 2u' + 401u = 401, u'(1) = -1, u'(1) = -6.872'$ using no-partitions ......................................................... 52

4.2 Lower and Upper bounds on the value of $8m_0 + 3m_1$ of the solution of ODE $u'' + 2u' + 401u = 401, u'(1) = -1, u'(1) = -6.872'$ using 4 partitions ......................................................... 52
Chapter 1

Introduction

Quantities that depend on space and/or time variables are governed by differential equations based on underlying physical principles. Partial Differential Equations (PDEs) not only accurately express these principles, but also help to predict the behavior of a system from an initial state of the system and given external influences. They are used to formulate and solve problems that involve unknown functions of several variables, such as the propagation of sound or heat, electrostatics, electrodynamics, fluid flow, elasticity, or more generally any process that is distributed in space, or space and time. Very different physical problems may have identical mathematical formulations. Hence, PDEs play an essential role in engineering problems.
1.1 Standard solution techniques for PDEs

Generally, partial differential equations can be represented as

\[ Lu = f \quad \text{in} \quad \Omega, \]

where

\( \Omega \) is the problem domain in which we are interested in solving,

\( L \) is a differential operator,

\( u \) is the unknown field variable in the domain \( \Omega \),

\( f \) is the forcing function.

In order to obtain a unique solution to the unknown field, boundary conditions have to be specified. A detailed description on applying boundary conditions can be found elsewhere. For example, the steady state heat conduction equation is given as,

\[ -\nabla \cdot (K \nabla T) = q \quad \text{in} \quad \Omega, \]

\[ T = T_0 \quad \text{on} \quad \partial \Omega_D, \]

\[ -K (\nabla T \cdot n) = q_0 \quad \text{on} \quad \partial \Omega_N, \]

where,

\( K \) is the thermal conductivity of the material,

\( T \) is the unknown temperature field we wish to compute,

\( q \) is the heat generated per unit volume,

\( \Omega \) is the domain in which we are interested in solving the problem

\( \partial \Omega_D \) is the Dirichlet boundary,
\( \partial \Omega_N \) is the Neumann boundary,

\( T_0 \) is the temperature field specified on Dirichlet boundary \( \partial \Omega_D \)

\( q_0 \) is the heat flux leaving/entering the

problem domain specified on Neumann boundary \( \partial \Omega_N \).

The governing differential equations with the boundary conditions specified on the
problem domain can be completely solved analytically/approximately depending on
the complexity of the problem.

The first approach towards solving these equations involves finding a analytical
solution of the equation over the whole domain. There are well established analyti-
cal methods for obtaining solutions for a wide variety of partial differential equations.
Usually, analytical solutions are available for partial differential equations over regular
domains (rectangular, circular) and some other simple cases. However, most engineer-
ing problems have complex geometric domains and nonlinear properties. Thereby, it
becomes imperative to rely increasingly on numerical techniques to solve PDEs in
order to understand and control the systems governed by them.

A few well established techniques for solving PDEs numerically are the Finite Dif-
fERENCE method, the Spectral method and the Finite Element method. The following
sections will give a brief description on these methods. There is a plethora of literature
available on the mathematical aspects of these methods, numerical implementation,
practical applications and limitations of these methods [1, 3, 4, 6, 7, 9, 10, 12, 13, 17, 20].

1.1.1 Finite difference method

The main idea of the finite difference method is to approximate the differential oper-
ator with a difference operator. The domain is discretized into grid points which can
be uniformly spaced (referred to as structured grids) or non uniformly spaced (un-
structured grids). The differential operator at each grid point is then expressed using
the values at the grid point and its neighbors, this is usually referred to as a stencil.
A stencil could be obtained using Taylor series expansion of the derivatives about a
grid point or by other well established techniques [7, 20]. For example, consider the following one dimensional problem,

\[ \frac{d^2u}{dx^2} = f \quad \forall \quad x \in [x_L, x_R], \]

and \( u|_{x=x_L} = 0, u|_{x=x_R} = 0 \) imposed as the boundary conditions. The domain is discretized into uniformly placed grid points \( x_i \) given by

\[ x_i = x_L + \frac{(x_R - x_L)(i - 1)}{10} \quad \forall \quad i \in [1..11]. \]

The finite difference approximation at each grid point \( x_i \) obtained using a central difference scheme would be

\[ \frac{u_{i+1} - 2u_i + u_{i-1}}{2\Delta x} = f_i \quad \forall \quad i \in [2..10]. \]

where \( u_i \) denotes the value of the unknown field at grid point \( i \), \( \Delta x = x_i - x_{i-1} \) and \( f_i \approx f \) an approximation to the actual forcing function. \( u_1 \) and \( u_{11} \) (the boundary points) are equal to zero (due to boundary conditions). The above set of simultaneous equations can be solved to obtain the unknown field at each grid point. It can be seen that the finite difference systems of equations is sparse in nature. This linear system can be solved in a variety of techniques such as Gaussian elimination, QR factorization, Conjugate gradient or even iterative schemes such as Jacobi, or Gauss-Seidel method [14].

1.1.2 Spectral methods

In spectral methods, the solution to the governing partial differential equation \( Lu = f \), is assumed to be approximated by a superposition of smooth global analytic functions [3, 4, 6, 9, 12],

\[ u(x) \approx u_N(x) = \sum_{n=0}^{N} a_n \phi_n(x), \]

18
where $u_N(x)$ is the best approximation to $u(x)$ in the space spanned by $\{\phi_0, \phi_1, ... \phi_N\}$. Subsequently, this is substituted into the governing equation and multiplied with a test function $v$; finally the residual is minimized over the entire domain.

$$\int_\Omega v(Lu - f) dA = 0 \quad \forall \ v \in X.$$

Based on the choice of $v$ (test function) and the interpolation scheme for $u$ (trial function), several numerical schemes can be obtained. Spectral methods use global basis functions in which the $\phi_n(x)$ is a polynomial (or trigonometric polynomial) of high degree which is non-zero, except at isolated points, over the entire computational domain. A Galerkin method would be obtained if the same expansion is used for the test function and the trial function [7, 17]

$$\int_\Omega \phi_m \left( \sum_{n=0}^{N} a_n \phi_n(x) \right) dA = \int_\Omega \phi_m f dA \quad \forall \ m \in 1 \ldots N,$$

$$a_n \int_\Omega \phi_m \left( \sum_{n=0}^{N} \phi_n(x) \right) dA = \int_\Omega \phi_m f dA \quad \forall \ m \in 1 \ldots N.$$

The above can be written in matrix form as

$$\sum_{n=1}^{N} L_{mn} a_n = f_m,$$

where $L_{mn} = \int_\Omega \phi_m L \phi_n(x) dA$, and $f_m = \int_\Omega \phi_m f dA$. The unknown's $a_n$ are obtained by solving the above set of linear equations. Typically in a Galerkin method, each of the test/trial functions are chosen so that the boundary conditions are automatically satisfied. Alternatively, the boundary conditions can be imposed as additional constraints to the linear equation; this class of methods is usually referred to as the Tau methods [12].

Test functions can be chosen as Dirac-delta functions at specific points in the domain. These classes of methods are usually referred as Collocation or Pseudospec-
tral methods and the points chosen are called the collocation points. The boundary
conditions can then be applied in the way it is done for Tau methods, as additional
constraints,
\[ \int_{\Omega} \delta(x - x_m)(Lu - f) \, dA = 0 \quad \forall \quad m \in \{1..N\}. \]
Collocation methods lead to a set of algebraic equations which are given as
\[ \sum_{n=1}^{N} L_{mn} a_n = f_m. \]
where \( L_{mn} = \int_{\Omega} L \phi_n(x) \, dA \) and \( f_m = \int_{\Omega} \phi_m f \, dA. \)

Usually spectral methods deploy polynomials of high order (typically Fourier series
or Chebyshev and Legendre polynomials), which gives high accuracy for a given \( N. \)
Although the generated equations result in full matrices, using fast iterative matrix
solvers these techniques can be more efficient than finite element or finite difference
methods for many classes of problems [3]. If the geometry of the problem is smooth
and regular, these techniques prove to be very useful.

1.1.3 Finite element method

Finite element method in concept is very similar to a Galerkin approach in Spectral
methods. The major difference is that this technique uses piecewise low-order polyno-
mial functions, instead of the high-order polynomial functions [1, 10, 13, 17]. Another
difference is that the finite element method relies on the weak form instead of using
the strong form of the equation. A weak form is obtained by integrating by parts, the
Galerkin weighted residual statement of the governing differential equation. There is
extensive literature available on mathematical aspects of finite element analysis and
applications that involve solving real-life problems.

There are several commercial software packages available that simulate a system
using finite element analysis. The major advantage of finite element analysis is that
it can handle complex geometries, non-linearity, interfaces and jumps in the problem
domain with little or no extra effort. The disadvantage is low accuracy (for a given
number of degrees of freedom $N$) because each basis function is a polynomial of low
degree $[10, 13]$.

### 1.2 Moment calculation using semidefinite optimization

Many times instead of an actual solution of a PDE, a particular functional of the
solution might be more of interest. For example we might be more concerned with
the average temperature along a physical boundary rather than the entire distribution
of temperature in a mechanical device. Therefore, under such circumstances solving
the PDE in the entire domain of the solution serves no purpose.

Recently, Bertsimas and Caramanis [2] proposed a technique that can be used
to calculate the moments of the solution based on optimization techniques without
calculating the actual solution. A plethora of information about a solution can be
obtained from the moments of the solution.

The key steps in the proposed scheme in [2] are:

- Write a set of linear equations involving the unknown moments (of the solution),
  and the boundary conditions. The number of equations generated are equal to
  the number of moments that we are interested in.

- Construct matrices using the unknown moments in a particular manner. These
  matrices are required to be positive semidefinite.

- Solve an optimization problem comprising of the linear constraints and the
  semidefinite constraints.

The accuracy of the moments obtained is enhanced by increasing the number of
linear equations. However, this will also increase the size of the semidefinite matrices;
hence the problem becomes bigger, which can lead to numerical instability. Thus,
we have proposed a modification to the methodology. A detailed description of the
existing methodology is presented later.
Structure of the paper

This thesis is structured as follows: In Section 2, we briefly summarize the proposed approach. In Chapter 3, we show the modified algorithm, and derive the formulation for the one-dimensional problem. Chapter 4 shows the numerical validation for the one-dimensional case. In Chapter 5, the two-dimensional formulation is presented, followed by numerical examples in Chapter 6. And lastly, Chapter 7, contains the concluding remarks.
Chapter 2

Description of the method

As mentioned earlier, this method is based on a modification of the technique proposed by Bertsimas and Caramanis [2]. Hence, a brief summary of the original algorithm is provided below, followed by a summary of the proposed technique.

2.1 Basic approach - theory

Given a problem domain \( \Omega \), and a linear PDE of the form

\[
Lu(x) = f(x), \quad x = (x_1, ... x_d) \in \Omega \subset \mathbb{R}^d,
\]

including the appropriate boundary conditions on \( \partial \Omega \), we wish to solve for some linear functional of the solution,

\[
\int Gu(x)dx.
\]

Henceforth, for convenience, we will simply write the equation as

\[
Lu = f.
\]

Since an operation on both sides of the equation should give us identical results, we can multiply by a given class of test functions \( D \) and integrate both sides of the...
equation as

\[ Lu = f \iff \int (Lu)\phi = \int (f)\phi, \quad \forall \phi \in \mathcal{D}. \]

We can choose a set of functions \( \mathcal{F} = \phi_1, \phi_2 \ldots \) that is a dense subset of \( \mathcal{D} \). Therefore, by linearity of integration we have

\[ Lu = f \iff \int (Lu)\phi = \int (f)\phi, \quad \forall \phi \in \mathcal{D} \]
\[ \iff \int (Lu)\phi_i = \int (f)\phi_i, \quad \forall \phi_i \in \mathcal{F}. \]

Although in [2], the authors have demonstrated the use of various classes of test functions, we shall be primarily concerned with monomials of the form \( x^a = x_1^{a_1}x_2^{a_2}\ldots x_d^{a_d} \).

### 2.2 Adjoint operator

The adjoint operator \( L^* \) is defined by the equation

\[ \int (Lu)\phi = \int u(L^*\phi), \quad \forall \phi \in \mathcal{D}. \]

If we have both \( L \) and \( L^* \), the equality in the original PDE becomes

\[ Lu = f \iff \int (Lu)\phi = \int (f)\phi, \quad \forall \phi \in \mathcal{D}, \]
\[ \iff \int (Lu)\phi_i = \int (f)\phi_i, \quad \forall \phi_i \in \mathcal{F}, \]
\[ \iff \int u(L^*\phi_i) = \int (f)\phi_i, \quad \forall \phi_i \in \mathcal{F}. \]

In the one-dimensional case, the general differential operator can be written as

\[ x^a \frac{\partial^b}{\partial x^b}. \]
Hence, the adjoint operator in the one-dimensional case

\[
\int_{\Omega} x^a (\partial^b u) \phi = \int_{\Omega} (\partial^b u)(x^a \phi) dx \\
= \int_{\Omega} (\partial^b u) \tilde{\phi} dx \\
= u^{b-1} \frac{\partial \tilde{\phi}}{\partial a} \bigg|_{\partial \Omega} + \cdots + (-1)^{k+1} u^{b-k} \frac{\partial^{(k-1)} \tilde{\phi}}{\partial a^{k-1}} \bigg|_{\partial \Omega} + \cdots \\
+ (-1)^{b+1} u \frac{\partial^{b-1} \tilde{\phi}}{\partial a^{b-1}} \bigg|_{\partial \Omega} + (-1)^b \int_{\Omega} u \frac{\partial^b \tilde{\phi}}{\partial a^b} dx.
\]

(2.1)

Here we have used the notation, \( \tilde{\phi} = x^a \phi \).

**Example.** Consider a test function \( \phi = x^2 \), and the differential operator

\[
x^a \frac{\partial^b u}{\partial x^b} = x^0 \frac{\partial^2 u}{\partial x^2} = u''.
\]

Hence,

\[
\int_{\Omega} x^a (\partial^b u) \phi = \int_{\Omega} (u'')(\phi) dx = \int_{\Omega} (u'')(x^2) dx.
\]

By applying the adjoint operator,

\[
\int_{\Omega} (u'')(\phi) dx = u' \phi \bigg|_{\partial \Omega} - u \phi' \bigg|_{\partial \Omega} + \int_{\Omega} u \phi'' dx \\
= u' x^2 \bigg|_{\partial \Omega} - 2ux \bigg|_{\partial \Omega} + 2 \int_{\Omega} u dx.
\]

Therefore, in the one-dimensional case, we have removed any differential operators from \( u(x) \), and converted it to a linear combination of the boundary values, and integrals of the solution over the whole domain. As we will see further, this adjoint operator is used extensively for generating linear constraints based on the differential equation.
2.3 Definitions

Defining a multi index,

\[ \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n), \]

such that,

\[ x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}. \]

Additionally,

\[ |\alpha| = \sum \alpha_i. \]

Furthermore, define \( x_{(d)} \) to be a vector of all monomials of degree less than or equal to \( d \) in lexicographic order. For example, if \( x = [x_1, x_2] \), and \( d = 2 \), then

\[ x_{(2)} = \begin{bmatrix} 1 & x_1 & x_2 & x_1 x_2 & x_3 \end{bmatrix}^T. \]

2.4 Moments and moment matrices

Defining \( u_0 \) as a lower bound of the solution \( u(x) \) such that

\[ u(x) \geq u_0 \quad \forall \quad x. \]

In many cases, \( u_0 \) is naturally known. For example, if \( u(x) \) represents temperature, then \( u(x) \geq 0 \). The integral of a monomial against the solution to the PDE \( u \) can be defined as

\[ m_\alpha = \int_{\Omega} x^\alpha (u(x) - u_0) dx = \int_{\Omega} x_1^{i_1} \cdots x_d^{i_d} (u(x) - u_0) dx. \quad (2.2) \]

Similarly we can also define the boundary moments of \( u \) along some portion of the boundary \( \partial \Omega \),

\[ z_\alpha = \int_{\partial \Omega} x^\alpha (u(x) - u_0) dx = \int_{\partial \Omega} x_1^{i_1} \cdots x_d^{i_d} (u(x) - u_0) dx. \]
We shall refer to \( m_\alpha \) and \( z_\alpha \) as the moments (of the solution), and boundary moments (of the solution) respectively, even though the solution \( u(\cdot) \) may not be a probability distribution. Note that in Equation 2.1 if \( \partial^k \phi \) are monomials of the form \( 1, x, x^2 \ldots \) these moment expressions are realized in the last term of the adjoint operator. Hence by selecting \( \phi_i \) as the family of monomials \( x^\alpha \), we can rewrite a differential operator in terms of its boundary values, and the variables \( m_\alpha \) and \( z_\alpha \) (i.e. moments of the solution).

For clarity, the moments will be subscripted with the individual powers of \( x \) in lexicographic order. For example, in the one-dimensional case,

\[
m_2 = \int_\Omega (u(x) - u_0) x^2 dx,
\]
while in the two-dimensional case,

\[
m_{4,2} = \int_\Omega (u(x_1, x_2) - u_0) x_1^4 x_2^2 dx_1 dx_2.
\]

**Moment matrices**

Given a set of monomials \( x_{(n)} \), we define the moment matrix

\[
M_{2n} = \int_\Omega (u(x) - u_0) x_{(n)} x_{(n)}^T dx. \tag{2.3}
\]

For example, in the one-dimensional case,

\[
M_{2n} = \begin{pmatrix}
m_0 & m_1 & \cdots & m_n \\
m_1 & m_2 & \cdots & m_{n+1} \\
\vdots & \ddots & \ddots & \vdots \\
m_n & & & m_{2n}
\end{pmatrix}.
\]

We also introduce the notation of the offset semidefinite matrix \( M_{2n+\alpha}, \alpha \in \{0, 1, \ldots\} \), which is created by multiplying \( (u(x) - u_0) x_{(n)} x_{(n)}^T \) in Equation 2.3 by the monomial at position \( \alpha \) in the vector \( x_{(n)} \) and then applying the integral operator to give a
new matrix. If $\alpha$ is greater than $\text{length}(\mathbf{x}(n))$, we will increase $n$ and add monomials to $\mathbf{x}$. For example, in the two-dimensional case,

$$M_{2n+1} = \begin{pmatrix}
m_{1,0} & m_{2,0} & m_{1,1} & \cdots \\
m_{2,0} & m_{3,0} & m_{2,1} & \cdots \\
m_{1,1} & m_{2,1} & m_{1,2} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}.$$ 

We can also restrict $u(\mathbf{x})$ to a boundary, and hence boundary moments can also be calculated.

### 2.5 Semidefinite constraints

It is clear that the matrices

$$\int_{\Omega} (u(\mathbf{x}) - u_0)\mathbf{x}_{(n)}\mathbf{x}_{(n)}^T d\mathbf{x}, \quad \int_{\partial \Omega} (u(\mathbf{x}) - u_0)\mathbf{x}_{(n)}\mathbf{x}_{(n)}^T d\mathbf{x},$$

are positive semidefinite for all $n$. In general, given a moment-like matrix that is positive semidefinite, it is not necessarily the moment matrix of some measure [5].

However, Bertsimas and Caramanis [2] have used a result by Schmüdgen [18] to demonstrate the following theorem, which provides the necessary and sufficient conditions for $M = m_\alpha$ and $z = z_\alpha$ to be valid moment sequences.

**Theorem 2.5.1.** [2, 18] Given $M = m_\alpha$ there exists a function $u(\mathbf{x}) \geq u_0$ such that

$$m_\alpha = \int_{\Omega} (u(\mathbf{x}) - u_0)\mathbf{x}^\alpha d\mathbf{x}$$

for all multi-indices $\alpha$

for a closed and bounded domain $\Omega$ of the form

$$\Omega = \{ \mathbf{x} \in \mathbb{R}^d : f_1(\mathbf{x}) \geq 0, \ldots, f_r(\mathbf{x}) \geq 0 \},$$

if and only if for all the subsets $I \subseteq \{1, \ldots, r\}$, and all $n \geq 0$, the matrix obtained
from the expression

\[ \int_{\Omega} (u(x) - u_0)x_nx_n^T \mathcal{I} \prod_{i \in \mathcal{I}} f_i(x) dx \]

by replacing \( \int (u(x) - u_0)x^\alpha \) by \( m_\alpha \) is positive semidefinite.

Hence, the semidefinite constraints applied are dependent on the domain boundary. Although the theorem requires all possible combinations of the boundary functions i.e. \( \prod_{i \in \mathcal{I}} f_i(x) \) to be imposed, in our experience, only a subset of the conditions need to be imposed as part of the optimization problem to give us good results.

Examples of semidefinite constraints

In the one-dimensional case, if the problem domain is \([-1, 1]\), we have \(-1 \leq x \leq 1\), which give the two boundary functions,

\[ f_1(x) = 1 + x, \quad f_2(x) = 1 - x. \]

Therefore, we get the constraints,

\[ \int_{\Omega} (u(x) - u_0)x_nx_n^T (1 + x) \, dx \succeq 0, \]
\[ \int_{\Omega} (u(x) - u_0)x_nx_n^T (1 - x) \, dx \succeq 0, \]

which translates to the semidefinite constraints,

\[
\begin{pmatrix}
  m_0 & m_1 & \cdots & m_n \\
  m_1 & m_2 & \cdots & m_{n+1} \\
  \vdots & \ddots & \ddots & \vdots \\
  m_n & m_{2n} & & \\
\end{pmatrix} \pm \begin{pmatrix}
  m_1 & m_2 & \cdots & m_{n+1} \\
  m_2 & m_3 & \cdots & m_{n+2} \\
  \vdots & \ddots & \ddots & \vdots \\
  m_{n+1} & m_{2n+1} & & \\
\end{pmatrix} \succeq 0.
\]
In the case of a two-dimensional problem in the domain $x_1, x_2 \in [-1, 1]$, there are four functions

$$f_1(x_1, x_2) = 1 + x_1, \quad f_2(x_1, x_2) = 1 - x_1,$$

$$f_3(x_1, x_2) = 1 + x_2, \quad f_4(x_1, x_2) = 1 - x_2.$$

In this case, the necessary and sufficient conditions [16] are

$$\int_{\Omega} (u(x) - u_0)x(n)x_1x_2(1 + x_1)(1 + x_2) \, dx \geq 0,$$

$$\int_{\Omega} (u(x) - u_0)x(n)x_1x_2(1 + x_1)(1 - x_2) \, dx \geq 0,$$

$$\int_{\Omega} (u(x) - u_0)x(n)x_1x_2(1 - x_1)(1 + x_2) \, dx \geq 0,$$

$$\int_{\Omega} (u(x) - u_0)x(n)x_1x_2(1 - x_1)(1 - x_2) \, dx \geq 0.$$

Each of the above conditions requires the sum/difference of four moment matrices be positive semidefinite. The offset position of each matrix is given by the position of the monomial in the vector $x(d)$. For example,

$$\int_{\Omega} (u(x) - u_0)x(n)x_2 \, dx = M_{2n+2},$$

because the position of $x_2$ in the vector $x(d) = 2$. Hence, we get the following semidefinite constraints,

$$M_{2n} + M_{2n+1} + M_{2n+2} + M_{2n+4} \geq 0,$$

$$M_{2n} + M_{2n+1} - M_{2n+2} - M_{2n+4} \geq 0,$$

$$M_{2n} - M_{2n+1} + M_{2n+2} - M_{2n+4} \geq 0,$$

$$M_{2n} - M_{2n+1} - M_{2n+2} + M_{2n+4} \geq 0.$$
Overall Formulation

If we are solving for certain functionals of the linear PDE,

\[ Lu = f, \]

the key steps are:

1. Compute the adjoint operator. Use the adjoint operator to write a linear equation in terms of boundary conditions, moments of the solution and test functions \( \phi \).

2. Generate the \( i \)th equality constraint \( 1 \leq i \leq n \) by replacing the \( \phi \) with test functions. In the cases illustrated later, these are monomials of the form \( x_1^{i_1} \ldots x_d^{i_d} \).

3. Generate semidefinite constraints among the moments that appear in the linear constraints. The nature of the semidefinite constraints is a feature of the boundary on which we want to provide support.

4. Compute upper and lower bounds on a particular moment by making it the objective in the optimization problem to be solved.

For example, in the optimization problem, one might minimize and maximize \( m_1 \) i.e. the first moment. As we increase \( N \) (the degree of the highest test function/monomial), the lower and upper bounds converge to each other. If the difference is small enough, the actual value of the moment can be inferred.

2.6 Motivation for modification

In simple examples, the algorithm performs quite well. However, when the solution \( u(x) \) is more complex, the gap between the lower and upper bounds might be large. Alternatively, one needs to generate many linear constraints before tight bounds can be obtained. However, this leads to the semidefinite problem getting bigger. Since the problems are being solved numerically, bigger problems can lead to numerical
instability. Bigger problems also mean more computation, and longer computational times. Hence, the motivation is to obtain the moments by solving smaller problems.

2.6.1 Proposed modification

The key steps in the modified scheme are as follows:

- Partition the domain into smaller sections. The number of partitions is an input to the algorithm.

- Translate each partition to the domain \([-1, 1]\) in each dimension. Mapping the original domain to a normalized coordinate system ensures numerical stability. This translation of domain will result in the coefficients of both \(L\) and \(f\) being changed. Each partition will have a different set of coefficients since the equations translating each partition will be different.

- Use the adjoint operator to write linear equations for each of the partitions.

- Introduce coupling conditions at the boundary between adjacent partitions; the boundary solution and derivative boundary conditions at a common boundary between two partitions are equal. These boundary conditions at the new boundaries will also be unknowns in the optimization problem.

- Write semidefinite constraints for each of the partitions as before. In this case, the domain for each partition is \([-1, 1]\) in each dimension.

- Solve the system of equations to obtain moments for each of the partitions. Using these, recover the original moments (these are linear combinations of the moments of the individual partitions).
Chapter 3

One dimensional case - ordinary differential equations (ODEs)

Ordinary Linear Differential equations have a very important role in physics. They model many real world applications nicely, and hence are used extensively in physics and engineering. However, with the exception of a few special cases, the only ODEs which have closed form solutions are those that are linear with constant coefficients. However, many ODEs have variable coefficients. To solve these, one resorts to approximation methods, including solutions in form of power series, Fourier series, and numerical methods [8]. Hence, a general form of the solution for linear ODEs has been developed here.

Recall that the ODE is simply a PDE in one dimension. Hence, this chapter will demonstrate the new methodology, by calculating bounds on moments of PDEs in the one-dimensional case. These are a natural step before progressing to higher dimensions. Moreover, most of the insights gained from the one-dimensional case will hold true for the two-dimensional case which is developed in Chapter 5.

Since we perform a domain translation in the algorithm, for clarity, let the independent variable in the original domain and the new domain be $s$ and $x$ respectively. Therefore, various aspects of the methodology will be illustrated by using a second-
order linear ODE which has the form

\[ p(s)u'' + q(s)u' + r(s)u = f(s), \]  
\[ (3.1) \]

in the original domain, and where

\[ p(s) = \sum_{i=0}^{n} a_i s^i, \quad q(s) = \sum_{i=0}^{n} b_i s^i, \quad r(s) = \sum_{i=0}^{n} c_i s^i. \]

However, all the results hold true for higher-order linear ODEs. We will demonstrate the algorithm at each step by the ODE

\[ s^2 u'' + 3u' + 3u = 0, \quad u'(0) = -2, \quad u'(1) = 2. \]  
\[ (3.2) \]

### 3.1 Partitioning and change of domain / domain mapping

The first step in this algorithm involves partitioning the domain into smaller pieces, and mapping each of these partitions to a new normalized domain. Let the number of partitions/divisions of the original domain be equal to \( D \) (Figure 3-1). Hence, each of the \( D \) partitions will be mapped into a new domain \([-1, 1]\). Let \( a_i \) and \( b_i \) be the coordinates of the beginning and end of the \( i \)th partition. The general domain transformation equation is given by

\[ x = -1 + \frac{2(s - a_i)}{b_i - a_i}, \]  
\[ (3.3) \]

Now differentiating with respect to \( s \),

\[ => \frac{dx}{ds} = \frac{2}{b_i - a_i} \]  
\[ (3.4) \]

\[ => \left( \frac{dx}{ds} \right)^n = \frac{2^n}{(b_i - a_i)^n}. \]  
\[ (3.5) \]

34
Figure 3-1: Partitioning the domain into $D$ partitions. Each partition is mapped onto a new domain $[-1, 1]$. The start and end of each partition are indicated by $a_i$ and $b_i$ respectively.

It should be noted that all the partitions are equal, and therefore,

$$b_i - a_i = \frac{b_D - a_1}{D} \quad \forall \quad i \in 1 \ldots D,$$

where $D$ is the number of partitions. Setting,

$$\kappa = 2 \times \frac{D}{b_D - a_1},$$

we get,

$$x = -1 + \kappa(s - a_i), \quad (3.6)$$

$$\Rightarrow \frac{dx}{ds} = \kappa, \quad (3.7)$$

$$\Rightarrow \left(\frac{dx}{ds}\right)^n = (\kappa)^n. \quad (3.8)$$
3.1.1 Scaling of constant coefficients

Consider the equation in the new domain, which can be rewritten as follows:

\[
p(x)u'' + q(x)u' + r(x)u = f(x)
\]

\[
\Rightarrow p(x)\frac{d^2u}{dx^2} + q(x)\frac{du}{dx} + r(x)u = f(x)
\]

\[
\Rightarrow p(s)\frac{d^2u}{ds^2} + q(s)\frac{du}{ds} + r(s)u = f(s)
\]

\[
\Rightarrow \kappa^2 p(s)\frac{d^2u}{ds^2} + \kappa q(s)\frac{du}{ds} + r(s)u = f(s)
\]

Therefore, the equation in the new boundary is just a scaled version of the equation in the old domain. Hence, without loss of generality, the modified equation in the new domain can be written as

\[
p(x)u'' + q(x)u' + r(x)u = f(x)
\] (3.9)

Any scaling of coefficients due to constant coefficients is identical for all the partitions. Note that the derivative conditions on the boundary also are scaled, while any boundary values remain unchanged.

Example. Consider the equation

\[
s^2 u'' + 3u' + 3u = 0,
\]

on the domain [0, 1]. If we are using two partitions (Figure 3-2), the domain transformation equation for Partition 1 is

\[
x = -1 + 4(s - 0),
\]

while the domain transformation equation for Partition 2 is

\[
x = -1 + 4(s - 0.5).
\]
Figure 3-2: Partitioning of the domain into two regions. The dotted line indicates the new boundary with the coupling conditions.
After scaling, the differential corresponding to both partitions is

\[ 16s^2u'' + 12u' + 3u = 0. \]  
(3.10)

### 3.1.2 Scaling of non-constant coefficients

When we perform a domain transformation, we have to replace every instance of the original independent variable i.e. \( s \) with the new variable (in the transformed domain). Rearranging Equation 3.6 for \( s \), we get

\[ s = \frac{x + 1}{\kappa} + a_i. \]  
(3.11)

Any instances of \( s \) in the differential equation (such as the \( s^2 \) term in the illustration ODE) have to be replaced by the right hand side term in Equation 3.11. Hence, in general any arbitrary power of the variable \( s^n \) and the general derivative term \( u^\text{p} \) can be rewritten as

\[ s^n \partial^\text{p} u = \left( \frac{1}{\kappa} \right)^n (x + 1 + \kappa a_i)^\text{p} u. \]

Let

\[ A = 1 + \kappa a_i, \]

\[ \Rightarrow s^n \partial^\text{p} u = \left( \frac{1}{\kappa} \right)^n (x + A)^\text{p} u. \]

From the binomial theorem,

\[ (x + A)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} A^k. \]
Therefore,

\[ s^n \partial^b u = \left( \frac{1}{\kappa} \right)^n \left( \sum_{k=0}^{n} \binom{n}{k} x^{n-k} A^k \right) \partial^b u. \]  

(3.12)

Hence, the additional terms are introduced in the polynomial corresponding to a particular differential operator. These new terms are linear transformations of the original terms.

**Example.** Partition 1 of the example: \( \kappa = 4, a_i = 0 \).

\[
16 s^2 u'' + 12u' + 3u = 16 \left( \frac{1}{4} \right)^2 (x + 1)^2 u'' + 12u' + 3u \\
= x^2 u'' + 2xu'' + 1u'' + 12u' + 3u.
\]

Partition 2 of the example: \( \kappa = 4, a_i = 0.5 \).

\[
16 s^2 u'' + 12u' + 3u = 16 \left( \frac{1}{4} \right)^2 (x + 3)^2 u'' + 12u' + 3u \\
= x^2 u'' + 6xu'' + 9u'' + 12u' + 3u.
\]

Hence, each of the partitions ends up with a different ordinary differential equation.

### 3.1.3 New boundary variables

Now, we will have to introduce additional boundary variables at the new boundaries between the partitions. For these variables, we will use the notation,

\[ u^b_d(\beta), \]

where \( d \in \{1 \ldots D\} \)

\[ \beta \in [k_1, k_2] \]

\[ b \in 0 \ldots \text{order of differential equation} - 1. \]
where $k_1$ and $k_2$ indicate the left and right boundary of a partition respectively (See Figure 3-2). Some of the new boundary conditions are unknown, and will remain so in the optimization problem. We will also scale any given derivative boundary conditions according to Equation 3.8.

**Example.** For the example, the new boundary conditions for the ODE in Equation 3.10 are

\[
\begin{align*}
    u_1(k_1) &= -1 & u_2(k_1) &= \text{??}, \\
    u_1(k_2) &= \text{??} & u_2(k_2) &= 1.
\end{align*}
\]

Note that the initial boundary conditions have been scaled appropriately.

### 3.2 Adjoint operator

The next step of the algorithm is to use the adjoint operator to simplify the equation

\[
\int_{k_1}^{k_2} (p(x)u'' + q(x)u' + r(x)u) \phi \, dx.
\]

Let us define the following terms:

\[
\begin{align*}
    \phi_p &= p(x)\phi, \\
    \phi_q &= q(x)\phi, \\
    \phi_r &= r(x)\phi.
\end{align*}
\]
Using the one-dimensional general formula for the adjoint operator, each of the terms can be simplified as follows:

\[
\int_{k_1}^{k_2} (p(x)u'')\phi dx = \int_{k_1}^{k_2} u''\phi dx
\]
\[= u'\phi'\big|_{k_1}^{k_2} - u\phi'_p\big|_{k_1}^{k_2} + \int_{k_1}^{k_2} u\phi'' dx. \quad (3.13)
\]

\[
\int_{k_1}^{k_2} (q(x)u')\phi dx = \int_{k_1}^{k_2} (u')\phi_q dx
\]
\[= u\phi_q\big|_{k_1}^{k_2} - \int_{k_1}^{k_2} u\phi'_q dx. \quad (3.14)
\]

\[
\int_{k_1}^{k_2} (r(x)\phi) dx = \int_{k_1}^{k_2} u\phi_r dx. \quad (3.15)
\]

Re-arranging, and combining the terms,

\[
(3.13) + (3.15) + (3.15) = u'\phi'\big|_{k_1}^{k_2} - u\phi'_p\big|_{k_1}^{k_2} + \int_{k_1}^{k_2} u\phi'' dx + \int_{k_1}^{k_2} (u\phi'' - u\phi'_q + u\phi_r) dx. \quad (3.16)
\]

Example. Using the scaled equation corresponding to Partition 1,

\[
\int_{k_1}^{k_2} ((x + 1)^2 + 12u_1' + 3u)\phi dx = u_1'((x + 1)^2\phi)\big|_{k_1}^{k_2} - u_1((x + 1)^2\phi)'\big|_{k_1}^{k_2} + u_1(12\phi)\big|_{k_1}^{k_2} + \int_{k_1}^{k_2} u((x + 1)^2\phi)'' - u(12\phi)' + u(3\phi) dx. \quad (3.17)
\]

### 3.3 Coupling conditions

There will be coupling conditions at the newly introduced boundaries between the partitions. The boundary solution values, and the boundary derivative values at the boundary should be equal for the two partitions on either side of the boundary.
Hence, we get the following general form of the coupling condition

\[ u_i^b(k_2) = u_{i+1}^b(k_1), \]

where \( i \in 1 \ldots D - 1 \)

\[ b \in 0 \ldots \text{order of differential equation} - 1. \]

The number of constraints added is a feature of the number of partitions, and the highest derivative operator in the differential equation.

**Example.** In the example we are using as illustration, these coupling conditions correspond to the boundary values being equal on the dotted line in Figure 3-2. Hence, the coupling constraints added to the optimization problem are

\[ u_1(k_2) = u_2(k_1), \]
\[ u_1(k_2)' = u_2(k_1)'. \]

### 3.4 Right hand side

The original form of the differential equation was

\[ Lu(s) = f(s). \]

Therefore, to generate the \( i^{th} \) equality constraint \( 1 \leq i \leq x \) we need

\[ \int u(L^* \phi_i) = \int f(s)\phi_i. \]

Since the left hand side has undergone a domain transformation, the right hand side also needs to go through a similar operation. Replacing the right hand side with Equation 3.11,

\[ \int_{k_1}^{k_2} f(s)\phi_i = \int_{k_1}^{k_2} f \left( \frac{x + 1}{\kappa} + a_i \right) \phi_i. \]
Special Case - Constant right hand side

Many differential equations have a constant right hand side i.e. \( f(x) = C \) where \( C \) is some constant. In this case, the right hand side simplifies to

\[
\int_{k_1}^{k_2} Cx^i dx = C \frac{x^{i+1}}{i+1}\bigg|_{k_1}^{k_2} \\
= C \left( \frac{(k_2)^{i+1}}{i+1} - \frac{(k_1)^{i+1}}{i+1} \right) \\
= \frac{C}{i+1} \left( (k_2)^{i+1} - (k_1)^{i+1} \right).
\]

3.5 Substituting \( \phi \)

Hence, for each partition, we will use the adjoint operator to give us a linear equation. Substituting the test functions of the form \( 1, x^1, x^2 \ldots x^N \) will give us \( N + 1 \) linear equations for each section/partition.

Example. Substituting the test functions in Equation 3.17, and equating to a zero right hand side, we get the equations noted below. All the moments in these equations correspond to Partition 1.

\( \phi = 1 \) ie \( x^0 \)

\[
(k_2 + 1)^2u'_1(k_2) + (k_1 + 1)^2u'_1(k_1) \\
-(2k_2 + 1)u_1(k_2) + (2k_1 + 1)u_1(k_1) \\
+12u_1(k_2) - 12u_1(k_1) \\
+ 5m_0 = 0
\]
$$\phi = x$$

$$k_2(k_2 + 1)^2 u_1'(k_2) - k_1(k_1 + 1)^2 u_1'(k_1)$$

$$-(3k_2^2 + 4k_2 + 1)u_1(k_2) + (3k_1^2 + 4k_1 + 1)u_1(k_1)$$

$$+12k_2u_1(k_2) - 12k_1u_1(k_1)$$

$$-8m_0 + 9m_1 = 0$$

$$\phi = x^N$$

$$(k_2^{N+2} + 2k_2^{N+1} + k_2^N)u_1'(k_2)$$

$$-(k_1^{N+2} + 2k_1^{N+1} + k_1^N)u_1'(k_1)$$

$$-((N + 1)k_2^{N+1} + 2(N + 1)k_2^N + Nk_2^{N-1})u_1(k_2)$$

$$+((N + 1)k_1^{N+1} + 2(N + 1)k_1^N + Nk_1^{N-1})u_1(k_1)$$

$$+12(k_2)^{N+1}u_1(k_2) - 12(k_1)^Nu_1(k_1)$$

$$+N(N - 1)m_{N-2} + (2(N + 1)N - 12(N - 1))m_{N-1}$$

$$+((N + 1)(N + 2) + 3)m_N = 0$$

### 3.6 SDP constraints

Each of the sectional moments will have their own semidefinite constraints. Since the new domain of each section is $[-1, 1]$, the following semidefinite constraints have to be satisfied on the domain.

$$\begin{pmatrix}
    m_0 & m_1 & \cdots & m_n \\
    m_1 & m_2 & \cdots & m_{n+1} \\
    \vdots & \ddots & \ddots & \vdots \\
    m_n & m_{2n} & & m_{2n}
\end{pmatrix} \begin{pmatrix}
    m_1 & m_2 & \cdots & m_{n+1} \\
    m_2 & m_3 & \cdots & m_{n+2} \\
    \vdots & \ddots & \ddots & \vdots \\
    m_{n+1} & m_{n+2} & & m_{2n+1}
\end{pmatrix} \succeq 0$$
3.7  Solving the system

Hence, we have a system of linear and semidefinite constraints. This optimization problem can be solved using any of the standard SDP solvers such as SeDuMi [19], SDPT3 [21] or any other standard SDP solver. YALMIP [11] is a MATLAB toolbox used for rapid prototyping of optimization problems. It provides a simple front end interface to many SDP solvers, and hence was used for rapid prototyping of the problems. Most of the problems were solved using SeDuMi 1.1 as the solver.

3.8  Recovering original moments

Recall that we are interested in the moments of the problem in the original domain. However, currently we have \( D \) set of moments, each of which corresponds to the moments of one of the sections. Therefore, using Equations 3.7 and 3.11, any arbitrary moment \( m_n \) for any partition between \([a_i, b_i]\) in the original domain \( s \) can be written as

\[
m_n = \int_{a_i}^{b_i} (s^a ds
\]

\[
= \int_{-1}^{1} f(x) \left( \frac{x + 1}{\kappa} + a_i \right)^n \frac{dx}{\kappa}
\]

\[
= \left( \frac{1}{\kappa} \right)^{n+1} \int_{-1}^{1} f(x) (x + 1 + \kappa a_i)^n dx
\]

\[
= \left( \frac{1}{\kappa} \right)^{n+1} \int_{-1}^{1} f(x) (x + A)^n dx
\]

Using the Binomial Theorem, this can be rewritten as
\[ m_n = \left( \frac{1}{\kappa} \right)^{n+1} \int_{-1}^{1} f(x) \sum_{k=0}^{n} \binom{n}{k} A^{n-k}(x)^k \]
\[ = \left( \frac{1}{\kappa} \right)^{n+1} \sum_{k=0}^{n} \binom{n}{k} A^{n-k} \int_{-1}^{1} f(x)x^k dx \]
\[ = \left( \frac{1}{\kappa} \right)^{n+1} \sum_{k=0}^{n} \binom{n}{k} A^{n-k} m_k \]

Note that \( m_n \) in Equation 3.18 is a moment in the original domain, while the \( m_k \) are moments in the transformed domain. Hence, the \( n \)th moment in the original domain is just a linear combination of all the moments from 0 to \( n \) in the transformed domain.

Using Equation 3.18, the moments of the original domain on each of the sections \([a_i, b_i]\) can be calculated. The moments on the full domain are obtained simply by summing up all the corresponding moments from each partition to obtain the full moments on the original domain.

### 3.9 Brief summary

Figure 3-3 provides a summary of the whole process. Consider the function on the top left corner. This is the function whose moments we are interested in. We partition the function domain into \( D \) partitions, and map each partition to a normalized scale. For each partition, we write linear, and semidefinite constraints. There are also coupling conditions between the partitions. We solve the optimization problem which results in \( D \) sets of moments that correspond to each partition. These are combined in a particular fashion to give us the moments of the original domain.

### 3.10 Higher order ODEs

The formulation above is easily expanded to higher dimensions as well. In higher dimensions, more boundary conditions are required to specify a particular solution of
Figure 3-3: Brief Summary of Algorithm - The original domain is partitioned and each partition is transformed to a normalized scale. Using the adjoint operator linear constraints are generated. Coupling conditions between partitions, and SDP constraints are also written and the optimization problem is solved. Finally, the original moments of the system are recovered by combining the moments corresponding to each partition.
the ODE, and therefore, there are more unknowns in the optimization problem.

3.11 Implementation for testing purposes

Let's consider the one-dimensional adjoint equation,

\[ \int_{\Omega} x^a (\partial^b u) \phi = \int_{\Omega} (\partial^b u) (x^a \phi) dx \]

\[ = \int_{\Omega} (\partial^b u) \phi dx \]

\[ = u^{b-1} \phi \bigg|_{\partial \Omega} + \ldots + (-1)^{k+1} u^{b-k} \phi^{(k-1)} \bigg|_{\partial \Omega} + \ldots \]

\[ + (-1)^{b+1} u \phi^{b-1} \bigg|_{\partial \Omega} + (-1)^b \int_{\Omega} u \partial^\phi \phi dx. \]

Given any derivative operator of order \( b \), and a coefficient in front, by applying the adjoint operator, it is decomposed into \( b \) boundary conditions, and one moment of the solution. The moment number is a function of the coefficient, the test function, and the order of the derivative operator; however, all these can be automated. Hence, instead of explicitly writing equations for any particular problem, a program was developed in Matlab to automatically generate the matrices that can be inputted into a SDP solver.

Methodology

The linear constraints corresponding to each partition can be written as

\[ A_i x_i = b_i, \quad \forall \quad i \in 1 \ldots D, \]

where \( A_i, x_i \) and \( b_i \) are generated dynamically as described below:

- \( x_i \) Vector: The \( x_i \) vector contains all the boundary conditions and the moments for the \( i \)th partition. The size of the vector is determined by the highest derivative operator in the differential equation. Some of the derivative conditions might be known, and therefore, there might be some known variables in
the vector. The general structure of $x_i$ is

\[
\begin{bmatrix}
    u_{b-1}(k_2) \\
    u_{b-1}(k_1) \\
    u_{b-2}(k_2) \\
    u_{b-2}(k_1) \\
    \vdots \\
    u'(k_2) \\
    u'(k_1) \\
    u(k_2) \\
    u(k_1) \\
    m_0 \\
    m_1 \\
    \vdots \\
    m_N \\
    \vdots \\
    m_{2N}
\end{bmatrix}
\]

- **$A_i$ Matrix**: The matrix, $A_i$, will contain the coefficients corresponding to the variables in the optimization problem for a particular partition. Each row of $A_i$ would correspond to a particular test function, $\phi$. Given a particular $\tilde{\phi}(x^b u^b)$, the coefficients in $A_i$ are calculated, and thus updated.

- **$b_i$ Vector**: The $b_i$ vector contains the terms $\int_{k_i}^{k_{i+1}} f \left( \frac{x+1}{\kappa} + a_i \right) \phi$ integrated with respect to $x$.

Additionally there will be coupling conditions between the partitions. Hence some variables in $x_i = x_{i+1}$. We would also generate the SDP constraints for each partition from the elements of $x_i$. 

49
Chapter 4

Numerical validation

The new methodology was applied to a few test cases. A wide range of ODEs were used as test examples. Some of the examples are simple ODEs, while others are more interesting such as the Bessel function, and Legendre polynomials. ODEs with known solutions were used as test cases. This allowed us to compute the actual moments analytically, which could then be compared with the solution from the optimization problem for accuracy.

4.1 Examples - plots and error plots

The objective of the solver in most of the one-dimensional cases was to minimize the value of $8m_0 + 3m_1$ of the solution of the ODE. This is an arbitrary function chosen for no particular reason. Recall that $m_0$ and $m_1$ are moments of the solution as defined in Equation 2.2.

By minimizing and maximizing the objective in the optimization problem, we would obtain a lower and upper bound for the objective function. For each problem, the bounds were initially calculated using no partitions and then using 2, 4, 6, 8 and 10 partitions. As well, the highest degree of the test functions $N$ was incrementally changed from 2 to 10 for all the test cases. For example, $N = 4$ implies that the test functions used for generating the linear constraints were $1, x, x^2, x^3$ and $x^4$.

Tables 4.1 and 4.2 show the lower and upper bounds calculated for objective of
the differential equation

\[ u'' + 2u' + 401u = 401, \quad u'(1) = -1, \quad u'(1) = -6.872, \]

using no-partitions and 4 partitions respectively.

Table 4.1: Lower and Upper bounds on the value of \( 8m_0 + 3m_1 \) of the solution of ODE \( u'' + 2u' + 401u = 401, \ u'(1) = -1, \ u'(1) = -6.872 \) using no-partitions

<table>
<thead>
<tr>
<th>N</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.58782387252627</td>
<td>331.2287657117187</td>
</tr>
<tr>
<td>4</td>
<td>8.60087181588462</td>
<td>12.41832038791917</td>
</tr>
<tr>
<td>5</td>
<td>9.351363490722703</td>
<td>11.31012965523155</td>
</tr>
<tr>
<td>6</td>
<td>9.35367056758392</td>
<td>10.13361730268152</td>
</tr>
<tr>
<td>7</td>
<td>9.41130776281981</td>
<td>10.00834412042844</td>
</tr>
<tr>
<td>8</td>
<td>9.49422948153400</td>
<td>9.83683698700403</td>
</tr>
<tr>
<td>9</td>
<td>9.49724559720386</td>
<td>9.80643867214983</td>
</tr>
<tr>
<td>10</td>
<td>9.55660706980515</td>
<td>9.80654282114558</td>
</tr>
</tbody>
</table>

In this case, the analytical value of the objective is 9.69401509901859. We see that by increasing \( N \), the lower bounds and upper bounds converge towards this value. The bounds obtained in the 4 partitions case is closer to the objective value, and the gap between them is smaller. Hence, this gap is an indication of the accuracy of the bounds computed. Lower the gap, better the estimate for the value of the objective function.

Therefore, Figures 4-1 to 4-7 show the plot of the solution obtained analytically, and the gap between the upper and lower bound for various test cases. In some cases, the

Table 4.2: Lower and Upper bounds on the value of \( 8m_0 + 3m_1 \) of the solution of ODE \( u'' + 2u' + 401u = 401, \ u'(1) = -1, \ u'(1) = -6.872 \) using 4 partitions

<table>
<thead>
<tr>
<th>N</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9.45099457753055</td>
<td>9.91766331785804</td>
</tr>
<tr>
<td>4</td>
<td>9.55343533597745</td>
<td>9.8123259136663</td>
</tr>
<tr>
<td>5</td>
<td>9.6092390097504</td>
<td>9.75148420145830</td>
</tr>
<tr>
<td>6</td>
<td>9.67541139308819</td>
<td>9.71141601527514</td>
</tr>
<tr>
<td>7</td>
<td>9.68830926142656</td>
<td>9.69884262144618</td>
</tr>
<tr>
<td>8</td>
<td>9.69332268823363</td>
<td>9.694744135362</td>
</tr>
<tr>
<td>9</td>
<td>9.6936826616983</td>
<td>9.6941784220312</td>
</tr>
<tr>
<td>10</td>
<td>9.69399830895136</td>
<td>9.69403139077513</td>
</tr>
</tbody>
</table>
no-partition case does not find an upper bound because the problem is unbounded. In such cases, the upper bound was set to be 1000 as this is substantially greater than the value of objective function.

4.1.1 ODEs with closed form solution

Figures 4-1 through to 4-5 demonstrate the application of the methodology on various ODEs with a closed form solution. Although simple, they give a very good insight on the behaviour of the algorithm.

4.1.2 ODEs with no closed form solution

Many ODEs do not have an explicit closed form solution. Rather, the solution is written as a summation of infinite terms. Figure 4-6 shows a practical application by finding the variance of the solution of the Bessel differential equation, $x^2 u'' + u' + (9x^2 - 1)u = 0$. The solution of this ODE is the $J_1(3x)$ function. The first and second moments were calculated, and the variance was calculated by $m_2 - m_1^2$. Figure 4-7 is a solution to the ODE $(x^2 - 4)u'' + 3u' + u = 0$. This is an interesting example, and the reasons for this are discussed in Section 4.2.1.

4.2 Discussion

4.2.1 Accuracy of solution

Consider the error plot in Figure 4-1. The gap between the lower and upper bound becomes small ($10^{-6}$) if we take a large value of $N$, even for the non-partition case. However, all the partition cases seem to have similar gaps ($10^{-10}$) between the upper and lower bounds of the solution. The solution of this ODE is a smooth function that is monotonically decreasing in the domain $[0, 1]$. In the case of the ODE, $u'' + 3u' - 4u = 0$, Figure 4-2 indicates that the non-partitioned case does not achieve as much accuracy as the various partitioning cases. Moreover, we see a little bit of segregation between the various partitioned cases, for lower values of $N$ used. The
Figure 4-1: Plot and error in the solution to ODE $u'' + 12u' + 50u = 4, u'(0) = 5, u'(1) = 0$
Figure 4-2: Plot and error in the solution to ODE $u'' + 3u' - 4u = 0$, $u'(0) = -1$, $u'(1) = 0$
Figure 4-3: Plot and error in the solution to ODE $u'' + 10u' + 50u = 24, u'(0) = 5, u'(1) = 0.5$
Figure 4-4: Plot and error in the solution to ODE $u'' + 2u' + 401u = 401, u'(0) = -1, u'(1) = -6.872$
Figure 4-5: Plot and error in the solution to a 5th order Legendre Polynomials. This example demonstrates that ODEs with polynomial solution do not benefit by partitioning the domain.
Solution to differential equation $x^2 u'' + x u' + (9x^2 - 1)u = 0 - \text{besselj}(1, 3x)$

Figure 4-6: Plot and error in the solution to ODE $x^2 u'' + u' + (9x^2 - 1)u = 0$, $u'(0) = 1.5, u'(1) = J_0(3) - J_2(3)$. Solution is the Bessel function $J_1(3x)$.
The solution to the ODE solved in Figure 4-3 is more complicated. The sign of the derivative changes twice in the domain. Here, the non-partition case achieves a gap of only $10^{-1}$ while the higher partitions reduce the gap between the lower and upper bounds to $10^{-6}$. Finally, if we observe the solution in Figure 4-4, there are three oscillatory peaks in the domain $[0,1]$. In this case, the increase in the accuracy of the non-partitioned case is very slow as we increase the degree of test functions used. Even with $N = 10$, the gap is approximately $10^{-1}$. In contrast, the gap is $10^{-9}$ when there are 10 partitions with $N = 10$. Moreover, simply by using only $N = 5$, all the partitioned cases perform as good or better than the non-partitioned case with $N = 10$. Clearly, there is an advantage in partitioning in this case, even the two partitions with lower values of $N$ perform better than no-partitions with higher degree test functions.

Figure 4-6 shows the gap in the variance of the solution to the Bessel differential equation. In this case, we see that all the partitions achieve gaps between the bounds of $10^{-8}$. It is interesting to note that the 8 and 10 partitions cases achieve this only
with $N = 5$; however we need more test functions to achieve the same level of accuracy for the non-partitioned, and the other partition cases. Hence, by partitioning we can reduce the number of linear equations, and the sizes of the semidefinite constraints.

Figure 4-7 is an interesting example. The solution is an infinite series summation

$$4 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{3}{32}x^4 + \frac{1}{30}x^5...$$

Since the decay in the coefficients is slow, to analytically calculate the moments, one would need about 100 terms to get an accurate answer. However, we can infer the moments by applying the proposed algorithm. In this case, we find that the partitioning cases have higher accuracy than the non-partitioning ones. Note that the moments can also be found by solving the ODE numerically.

Hence, partitioning the domain definitely increases the accuracy of the solution in all of these cases mentioned. The increase in accuracy seems to be correlated with the complexity of the function in the domain. Here the partition increases accuracy greatly if the function has lots of variations (oscillations) in the domain.

However, Figure 4-5 shows the results for the Legendre polynomial of order 5. The corresponding differential equation is

$$(1 - x^2)u'' - 2xu' + 20u = 30, u'(0) = 0, u(0) = 2.5,$$

while the solution is

$$\frac{5}{2} - 10x^2 + \frac{35}{3}x^4.$$ 

Surprisingly, the non-partitioning case achieves high accuracy even with the degree of test function equal to 3. (gap of $10^{-10}$). Moreover for all cases, increasing the number of moments reduces the accuracy. This reduction can be attributed to numerical errors. However, this example demonstrates an important point which was observed during testing. For ODEs that have a polynomial solution, the non-partition case performs excellently; as well, partitioning increases the error because of numerical reasons. Hence, partitioning is only of benefit if there is a large gap between the
lower and upper bounds of the non-partitioning case.

4.2.2 Running time

One question which naturally arises is the time taken to solve these problems. The partitioned problems are bigger, and should take more time. Is the additional accuracy worth the additional run time?

However, getting a correct measure of the time taken was hard. It was found that the same code would have different running times at different executions. YALMIP and SeDuMi are both written in Matlab, which became slow when the code was executed repeatedly due to memory issues. Moreover, the code was written solely for the purpose of validating the algorithm, and minimizing run time or memory utilization was not a criteria. For example, many for loops were used in the code, which are not ideal in Matlab.

Figure 4-8 shows the time taken to solve the problem illustrated in Figure 4-4. This example had demonstrated enormous benefits in partitioning the domain. YALMIP reported both the time to required to construct the problem, and the time taken by the solver to solve it. In general, the time taken by YALMIP to construct
a problem is much smaller than the solver time, and therefore, only the solver time has been used.

It is interesting to note that the case with two partitions has lower execution time than the original methodology. This can be possible due to greater numerical stability achieved as a result of the normalized scale. To get the same accuracy with no partitions and \( N = 10 \), we only need 2 partitions and \( N = 6 \), which takes half the time to execute. If we want additional four decimal places of accuracy, we can partition the domain into four, with \( N = 10 \). This takes about 80% of the original case's time. The case with 10 partitions \((N = 10)\) takes about 25% more time to execute, and gives a higher accuracy of 8 decimal. Hence, in such cases where the non-partition methodology has a high gap between the upper and lower bounds, partitioning certainly has enormous advantages, both in accuracy and running time.

### 4.2.3 Inferring the solution

![Graph showing the solution, and the bounds on \( m_0 \) for the ODE \( u_x x + 3u_x - 4u = 0, u_x(0) = -1, u_x(1) = 0 \) with no partitions. The average value does not indicate very well what is happening in the domain.](image)

The value of \( m_0 \) gives the average value of the solution if its normalized. Therefore another advantage of partitioning is that it gives us a better picture idea of what is
Solution to differential equation: \( u_{xx} + 3u_x - 4u = 0, u_x(0) = -1, u_x(1) = 0 \)

and lower and upper bounds of \( m_0 \) using 4 partitions.

Figure 4-10: Graph showing the solution, and the bounds on \( m_0 \) for the ODE \( u_{xx} + 3u_x - 4u = 0, u_x(0) = -1, u_x(1) = 0 \) with 4 partitions. The average values of the four partitions indicate better what is happening in the domain.

happening to the solution in the domain. Consider Figure 4-9. It shows the plot of the ODE

\[ u_{xx} + 3u_x - 4u = 0, u_x(0) = -1, u_x(1) = 0, \]

and the lower and upper bounds of \( m_0 \) using \( N = 10 \). Since the gap between the upper and lower bounds is small, we do not see a gap. This average value gives us a poor indication of what is happening to the function.

Now consider Figure 4-10 which plots the ODE, and the lower and upper bounds when the domain is partitioned into four. We obtain four sets of bounds, each corresponding to the appropriate partition. In this case, we can see that the average value of \( m_0 \) is different for the four partitions. The four values approximate the solution better than the value obtained from no partitions. Had we partitioned the domain more, we would have better approximations of the average value of the solution.

Hence, partitioning has the additional advantage of giving us a better picture of what is happening to the solution in the domain.
4.2.4 Summary

The key points to be taken from the examples presented are:

- Partitioning certainly increases the accuracy of the bounds obtained. In some examples, the partitioning increased accuracy by 9 decimal places.

- The increase in accuracy of the partitioning seems to be related to the variation/oscillation of the solution in the domain. In the cases where there are lots of variation partitioning, greatly enhances the bounds achieved.

- In the case of ODEs with polynomial solutions, the no-partition case achieves smaller gaps between the bounds. Under such circumstances, partitioning only worsens the bounds achieved. Partitioning is only beneficial when the gap between lower and upper bounds in non-partitioning case is large.

- The time taken to solve the partitioned problem is less for smaller number of partitions due to numerical stability. In addition, the benefits of obtaining higher accuracy outweighs the longer run times required for the solving the problem with higher number of partitions.

- Partitioning the solution, and using the normalized value of $m_0$ gives us an indication of what is happening to the solution in the problem domain.
Chapter 5

Second order two-dimensional partial differential equations

Although the one-dimensional problems have very promising results, existing schemes for solving them are extremely well developed. A wide range of analytical and numerical schemes already exist for the ODE schemes. In contrast, many two-dimensional partial differential equations (PDE) are typically solved numerically. Moreover, whole range of interesting problems can only be described using two-dimensional PDEs. Therefore, it was natural to attempt to evaluate the technique to two-dimensional problems.

The two-dimensional case is very similar to the one-dimensional case, and hence the following sections explain the two-dimensional case briefly, and highlight the major differences. As before, we will be partitioning the domain and transforming each partition to a normalized domain. Therefore, we will denote the independent variables in the original domain by \((s, t)\). Now we are considering a two-dimensional linear, non-homogeneous, second order partial differential equation in two variables with non-constant coefficients, the general form of which can be written as

\[
p_1(s, t)u_{ss} + p_2(s, t)u_{tt} + p_3(s, y)(s, t)u_{st} + q_1(s, t)u_s + q_2(s, t)u_t + r(s, t)u = f(s, t).
\]
5.1 Partitioning and domain transformation

In the two-dimensional case, we assume for convenience that the domain is square, hence the domain is equal in both dimensions. The domain is partitioned into squares each of which is transformed on to the square $[-1,1] \times [-1,1]$.

Let $a_s, a_t, b_s, b_t$ correspond to the coordinates of the four corners of the particular partition being transformed. Consider Figure 5-1 where we are partitioning the original domain into four smaller partitions. Each of these squares will be transformed to the domain $[-1,1] \times [-1,1]$. Therefore each will have its own constants depending on its location. For e.g. the square marked (1,1) will have $(a_s, a_t, b_s, b_t) \to (0, 0, \frac{1}{2}, \frac{1}{2})$, while the square marked (2,2) will have $(a_s, a_t, b_s, b_t) \to \left(\frac{1}{2}, \frac{1}{2}, 1, 1\right)$. It should be noted that we have dropped the subscript corresponding to the partition number for clarity. Had we kept the partition numbers, we would have two numbers, one for each dimension.

Since we are assuming that the original domain is equal in both dimensions, let

$$\kappa = \frac{2}{b_s - a_s} = \frac{2}{b_t - a_t}.$$  

There will be two transformation equations corresponding to each dimension. Let the new independent variables be denoted by $(x, y)$. The domain transformation equation for the two dimensions are

$$x = -1 + 2 \times \frac{s - a_s}{b_s - a_s}$$

$$\Rightarrow \frac{\partial x}{\partial s} = \frac{2}{b_s - a_s}$$

$$\Rightarrow \frac{\partial x}{\partial t} = \kappa$$

$$\Rightarrow \left(\frac{\partial x}{\partial s}\right)^2 = \kappa^2.$$
Figure 5-1: Sketch of the partition of the two-dimensional region. The arrows on the dotted line indicate the coupling conditions between partitions.
t - a_t)
\Rightarrow \frac{\partial y}{\partial t} = \frac{2}{b_t - a_t}
\Rightarrow \frac{\partial y}{\partial t} = \kappa
\Rightarrow \left(\frac{\partial y}{\partial t}\right)^2 = \kappa^2.

5.1.1 Constant coefficients

As before, any constant coefficients will be scaled due to the domain transformation. Without loss of generality, after scaling, the partial differential equation corresponding to any partition in the new domain can be written as

\[ p_1(x, y)u_{xx} + p_2(x, y)u_{yy} + p_3(x, y)u_{xy} + q_1(x, y)u_x + q_2(x, y)u_y + r(x, y)u = f(x, y). \]

5.1.2 Non-constant coefficients

As in the one-dimensional case, any non-constant coefficients containing \( s \) or \( t \) will result in the cascading effect of modifying the lower powers in the polynomial corresponding to that particular differential operator. In this case, the general term is
given by $s^t \partial^b u$. Setting,

$$s = \left( \frac{x + 1}{\kappa} + a_s \right)$$

$$= \frac{1}{\kappa} (x + A),$$

where $A = 1 + \kappa a_s$,

and

$$t = \left( \frac{y + 1}{\kappa} + a_t \right)$$

$$= \frac{1}{\kappa} (y + B),$$

where $B = 1 + (\kappa)a_t$,

and substituting,

$$s^t m \partial u = \left( \frac{x + A}{\kappa} \right)^m \left( \frac{y + B}{\kappa} \right)^n \partial^b u$$

$$= \sum_{j=0}^{m+n} \sum_{k=0}^{n} \binom{m}{j} \binom{n}{k} x^{n-k} A^k y^{m-j} B^j \partial^b u.$$

### 5.2 Adjoint Operator

Unlike the one-dimensional case, where there were fixed number of boundary variables (boundary values, and derivative boundary values), in two dimensions, there are $O(N)$ boundary variables. Therefore, instead of boundary variables, the linear constraints will contain boundary moments. To compute the adjoint operator, we use Green's theorem,

$$\int_C Ldx + Mdy = \int_D \left( \frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} \right) dA.$$
Considering the term \( u_{xx} \phi \), the derivation of the adjoint operator is presented below.

\[
\therefore \frac{\partial (u_x \phi)}{\partial x} = u_{xx} \phi + u_x \phi_x
\]

\[
\Rightarrow \frac{\partial (u_x \phi)}{\partial x} - u_x \phi_x = u_{xx} \phi
\]

\[
\Rightarrow \int \frac{\partial (u_x \phi)}{\partial x} \, dx \, dy - \int u_x \phi_x \, dx \, dy = \int u_{xx} \phi \, dx \, dy.
\]  

(5.1)

By using Green's Theorem, we can rewrite Equation 5.1 as

\[
\int_{\partial \Omega} u_x \phi \, dy \, dS - \int u_x \phi_x \, dx \, dy = \int \Omega u_{xx} \phi \, dx \, dy
\]

\[
\Rightarrow \int \Omega u_{xx} \phi \, dx \, dy = - \int u_x \phi_x \, dx \, dy + \int_{\partial \Omega} u_x \phi \, dy \, dS
\]

\[
\Rightarrow \int \Omega u_{xx} \phi \, dx \, dy = - \int u_x \phi_x \, dx \, dy + \int_{\partial \Omega} u_x \phi n_x \, dS.
\]  

(5.2)

Similarly,

\[
\therefore \frac{\partial (u \phi_x)}{\partial x} = u_x \phi_x + u \phi_{xx}
\]

\[
\Rightarrow \int \frac{\partial (u \phi_x)}{\partial x} \, dx \, dy = \int u_x \phi_x \, dx \, dy + \int u \phi_{xx} \, dx \, dy
\]

\[
\Rightarrow \int \Omega u_x \phi_x \, dx \, dy = - \int u \phi_{xx} \, dx \, dy + \int \frac{\partial (u \phi_x)}{\partial x} \, dx \, dy
\]

\[
\Rightarrow \int \Omega u_x \phi_x \, dx \, dy = - \int u \phi_{xx} \, dx \, dy + \int_{\partial \Omega} u \phi_x \, dy \, dS
\]

\[
\Rightarrow \int \Omega u_x \phi_x \, dx \, dy = - \int u \phi_{xx} \, dx \, dy + \int_{\partial \Omega} u \phi_x n_x \, dS.
\]

Substituting this in Equation 5.2, we get

\[
\int \Omega u_{xx} \phi \, dx \, dy = \int \Omega u \phi_x \, dx \, dy + \int_{\partial \Omega} u_x \phi n_x \, dS - \int_{\partial \Omega} u \phi_x n_x \, dS.
\]

The boundary \( \partial \Omega \) is oriented to have unit outward normal \((n_x, n_y)\). Therefore, for a
square domain on the boundaries \([k_1, k_2] \times [k_1, k_2]\),

\[
\int_{\partial \Omega} u_\omega \phi n_\omega dS = \int_{k_1}^{k_2} u_\omega(x = k_2, y)\phi(x = k_2, y) - u_\omega(x = k_1, y)\phi(x = k_1, y) dy.
\]

Hence,

\[
\int u_{zz}\phi dxdy = \int_{\Omega} \frac{\partial^2 u}{\partial x^2} \phi dxdy
\]

\[
= \int_{\Omega} u\phi_{zz} dxdy
\]

\[
+ \int_{k_1}^{k_2} u(x = k_2, y)\phi(x = k_2, y) dy
\]

\[- \int_{k_1}^{k_2} u(x = k_1, y)\phi(x = k_1, y) dy
\]

\[- \int_{k_1}^{k_2} u(x = k_2, y)\phi(x = k_2, y) dy
\]

\[+ \int_{k_1}^{k_2} u(x = k_1, y)\phi(x = k_1, y) dy. \tag{5.3}\]

Similarly, equations can be generated for all the six terms in the general differential equation. The set of all equations have been presented in Appendix A.

5.2.1 Moment variables

As Equation 5.3 demonstrates that applying operator results in a linear equation containing the moments of the solution, and boundary moments.

Let us define

\[
m_{i,j} = \int_{k_1}^{k_2} \int_{k_1}^{k_2} x^iy^j u(x, y) dxdy.
\]

If we restrict the solution to one of the boundaries, we can define the boundary moment along a particular boundary as follows:

\[
b_{k_2}^{x} := \int_{k_1}^{k_2} u(x = k_2, y)y^j dy
\]

\[
b_{k_2}^{y} := \int_{k_1}^{k_2} u(x, y = k_2)x^i dx
\]

\[
b_{k_1}^{x} := \int_{k_1}^{k_2} u(x = k_1, y)y^j dy
\]

\[
b_{k_1}^{y} := \int_{k_1}^{k_2} u(x, y = k_1)x^i dy
\]
Similarly, moments can also be defined for the moments along the boundary for the derivative functions. However, the derivative of the function can be taken with respect to $x$ or $y$. Hence there are two sets of similar derivative moments.

Taking the derivative of the function with respect to $x$ i.e. \( \frac{\partial u}{\partial x} \)

\[
\begin{align*}
    d_{y=1}^{x=k_2} &:= \int_{k_1}^{k_2} u_x(x = k_2, y) y^i \, dy \\
    d_{x=1}^{x=k_2} &:= \int_{k_1}^{k_2} u_x(x = k_2, y) x^i \, dy
\end{align*}
\]

Now differentiating with respect to $y$ i.e. \( \frac{\partial u}{\partial y} \)

\[
\begin{align*}
    d_{y=1}^{y=k_2} &:= \int_{k_1}^{k_2} u_y(x, y = k_2) y^i \, dx \\
    d_{x=1}^{y=k_2} &:= \int_{k_1}^{k_2} u_y(x, y = k_2) x^i \, dx
\end{align*}
\]

5.3 Calculating boundary and derivative boundary moments in the new domain and coupling conditions

As mentioned earlier, in the two-dimensional case, the linear equations contain boundary moments instead of boundary conditions. Typically, in a PDE problem the boundary conditions are provided as constants or functions (for e.g. $u(0, y) = 1 - y$). The given boundary conditions can describe the solution, or the derivatives of the solution at the boundary. The conditions provided are a feature of the problem, and sometimes can be a mixture of the two on various boundaries of the problem domain. Using these functions, we can calculate the moments on the boundary. In the one-dimensional case, the boundary moments were simply scaled according to the number of partitions. However, in the two-dimensional case, the moments at the boundary of the new domain are linear combinations of the moments of the original domain.
Consider Figure 5-1. Let the boundary condition provided at $u|_{s=0} = (1 - t)$. Hence the boundary moment for sections (1,1) and (2,1) on the original domain are calculated as

Moments for section (1,1) = $\int_{0}^{1/2} (1 - t)t'dt$,

Moments for section (1,2) = $\int_{1/2}^{1} (1 - t)t'dt$.

Now each square is shifted to the domain $[-1, 1] \times [-1, 1]$. Since,

$$\int_{a_t}^{b_t} f(t)t'dt = \int_{-1}^{1} f(y) \left( \frac{y}{\kappa} + \frac{1}{\kappa} + a_t \right) \frac{dy}{\kappa} \quad (5.4)$$

the moments on the new domain can be obtained by rearranging Equation 5.4, and using the binomial theorem to expand the moment expressions to give values of $\int_{-1}^{1} f(y)y^i dy = \int_{-1}^{1} b^{x=0}_{y=i}$ for various $i$ on both the partitions.

5.3.1 Coupling conditions

In the two-dimensional case, each partition shares a new boundary with at least two other partitions. If the number of partitions is greater than 2, the partitions in the middle of the domain will have coupling conditions with all four neighbors. Using the notation

$$b^{x=k_2}_{x=k_1}(m, n) = \int_{k_1}^{k_2} u(x = k_2, y)y^i dy \quad \text{corresponding to section}(m, n)$$

we can write the following coupling conditions when $D > 1$ (i.e. the original domain is partitioned).

$$\begin{align*}
\forall \ m \in 1 \ldots D - 1 \\
\forall \ n \in 1 \ldots D \n\end{align*}$$

75
\[
\begin{align*}
\beta^y_i(m, n) &= \beta^y_i(m, n+1) \quad \forall \ n \in 1 \ldots D - 1 \\
\delta^x_i(m, n) &= \delta^x_i(m, n+1) \quad m \in 1 \ldots D \\
\delta^y_i(m, n) &= \delta^y_i(m, n+1)
\end{align*}
\]

As earlier, the boundary moments of the boundaries in the middle of the original domain are unknowns in the optimization problem.

### 5.4 Right hand side

The right hand side is calculated as it is done for the one-dimensional case, i.e. by replacing the variables \( s \) and \( t \), and integrating for various test functions.

### 5.5 Substituting \( \phi \)

In this case, monomials of the form \( x^i y^j \) are used as test functions to give us linear equations involving derivative moments, boundary solution moments, and moments of the solutions. The moments of the solutions are unknowns. Additionally, one of the derivative moments or the boundary moments are unknowns depending on how the initial conditions were specified.
5.6 SDP constraints

5.6.1 Moment matrices

Since the domain of each partition is \([-1, 1] \times [-1, 1]\), the semidefinite constraints imposed on the solution in the optimization problem for each partition are

\[
M_{2n} + M_{2n+1} + M_{2n+2} + M_{2n+4} \succeq 0,
\]

\[
M_{2n} + M_{2n+1} - M_{2n+2} - M_{2n+4} \succeq 0,
\]

\[
M_{2n} - M_{2n+1} + M_{2n+2} - M_{2n+4} \succeq 0,
\]

\[
M_{2n} - M_{2n+1} - M_{2n+2} + M_{2n+4} \succeq 0.
\]

The size of the matrices is dependent on the number of test functions used.

5.6.2 Boundary moments

In this case, we have to impose semidefinite constraints on the boundary moments, both on the solution and on the derivative moments. The domain of the boundary is \([-1, 1]\). Hence the following semidefinite constraints imposed are

\[
\begin{pmatrix}
z_0 & z_1 & \cdots & z_n \\
z_1 & z_2 & \cdots & z_{n+1} \\
\vdots & \ddots & \vdots & \vdots \\
z_n & z_{n+1} & \cdots & z_{2n}
\end{pmatrix}
\pm
\begin{pmatrix}
z_1 & z_2 & \cdots & z_{n+1} \\
z_2 & z_3 & \cdots & z_{n+2} \\
\vdots & \ddots & \vdots & \vdots \\
z_{n+1} & z_{n+2} & \cdots & z_{2n+1}
\end{pmatrix} \succeq 0
\]

where \(z_i\) is one of the boundary moments specified in Section 5.2.1. These constraints are applied to all boundary moments which are unknown in the optimization problem.

It should be noted that these boundary moments are extremely critical to obtain an upper bound on the value of a moment. Without them, when we are maximizing any moment, the objective function is unbounded.
5.7 Recovering the moments

The optimization problem will return the moment on the new domain for each of the partitions. These have to be scaled back to recover the moments on the original partitions. They are a linear combination of the moments on the $[-1, 1] \times [-1, 1]$ domain. Hence to recover them, we can simply use the formula

$$m_{n,m} = \left( \frac{1}{\kappa} \right)^{m+n+2} \sum_{j=0}^{m} \sum_{k=0}^{n} \binom{m}{j} \binom{n}{k} (1 + \kappa \alpha)^{n-k} (1 + \kappa \alpha)^{m-j} \mu_{k,j} \quad (5.5)$$

Equation 5.5 can easily be derived in fashion similar to the one-dimensional case. Once all the moments corresponding to each section is obtained, they can be summed up to give the moment on the original domain.

5.8 Implementation

The implementation for the two-dimensional case was similar to the one-dimensional case. The data structures had to be modified to accommodate the two-dimensional nature of the partitions.
Chapter 6

Numerical validation

Similar to the one-dimensional case, different two-dimensional PDEs were used to validate the methodology. In this case, different objective functions were used for different problems. For each objective, the lower and upper bounds have been calculated, and the gap has been plotted. The lower the gap between the values, the better we can estimate the value of the objective function.

In each case, the bound was initially calculated using no-partitions and then using 2, 3, 4 and 5 partitions in each dimension. Different degrees of test functions were used for different problems. In this case, $N$ reflects the highest degree of monomial used in the test functions. Hence, $N = 4$ will use the monomials $1, x, y, x^2, y^2, xy, x^3, y^3, x^4, y^4$. This is the same as $x = [x, y]$ and the test functions are the monomials in $x^{(4)}$. It should be noted that the increase in the number of test functions is not linear when $N$ is increased. For example, when $N = 4$, the number of test functions is 15, while when $N = 5$, the number of test functions = 21. For all cases, we have found bounds on

$$m_{i,j} = \int_{\Omega} u(x,y)x^iy^j dxdy$$

for various values of $i$ and $j$. 

79
6.1 Two-dimensional PDEs

We present four two-dimensional examples over the domain \( \Omega = [0, 1]^2 \).

6.1.1 Poisson's equation

The partial differential equation of the Poisson’s Equation is

\[
\Delta u = f(x, y).
\]

Setting the right hand side function, and boundary condition appropriately, we obtain a solution as

\[
u(x, y) = \sinh(x) \cosh(x) + \cosh(y) \sinh(y).
\]

Figure 6-1: Gap between lower bound and upper bound for \( m_{1,0} \) of the differential equation \( \Delta u = 4 \sinh(x) \cosh(x) + 4 \cosh(y) \sinh(y) \)
6.1.2 Helmholtz equation

The Helmholtz Equation has the form

\[ \Delta u + k^2 u = f(x, y). \]

Setting \( k = 1 \), and right hand side function and boundary condition appropriately, we obtain the solution

\[ u(x, y) = e^{x+y}. \]

---

Figure 6-2: Gap between lower bound and upper bound for \( m_{0,0} \) of the differential equation \( \Delta u + u = 3e^{x+y} \)
6.1.3 Non-constant coefficient example

The governing PDE for this example is

\[ u_y = x^4 u_{xx} + u, \]

where, the boundary conditions have been specified by specifying the boundary moments, such that the corresponding solution is

\[ u(x, y) = xe^y. \]

6.1.4 Heat equation

Although we have been using second order elliptic differential equations, there is no reason why we cannot choose parabolic or hyperbolic equations. As well, although the presentation of the boundary conditions may not be natural for these kind of problems, the methodology can still be applied.

\[ u_y = u_{xx} + u, \]

which describes one-dimensional mass transfer in a quiescent medium with a first-order volume chemical reaction, or a heat transfer in a one-dimensional rod with ambient medium having a constant temperature [15]. By choosing appropriate boundary conditions, we have the solution as,

\[ e^y \text{erf} \left( \frac{x}{2(y + 1)^{0.5}} + e^y \right). \]
6.2 Discussion

6.2.1 Accuracy for lower degree test functions

Figure 6-1 shows the gap between the lower and upper bounds for the value of $m_{1,0}$ for the Poisson’s Equation. In this case, we observe that the non-partitioned case has an error of $10^{-2}$ when setting $N = 5$, while all the partitioned cases reduce the error to approximately $10^{-4}$ for the same degree of test function. Moreover, we can obtain the same error as the non-partition, $N = 5$ case by setting $N = 2$ only for the cases with 3, 4 or 5 partitions. Hence, partitioning does give a slight increase in accuracy with lower test functions. Moreover, if we take the middle point of the gap between the lower and upper bound as the inferred value of our moment, the error is less than '0.005%'.

In Figure 6-2, the bounds on value of $m_{0,0}$ of the Helmholtz Equation is calculated. Similar to the Poisson’s equation, we see the gap decrease to $\approx 10^{-4}$, when we
partition the domain into 5 partitions. Additionally, by using 5 partitions and setting $N = 2$, we obtain similar result to no-partitioning and $N = 5$.

In both these cases, we certainly observe an increase in the accuracy of the solution. Moreover, we can obtain the same accuracy as the non-partitioning case by using less number of test functions, and higher number of partitions. However, the increase in the accuracy is not as dramatic as that seen in some of the one-dimensional examples.

6.2.2 Numerical problems

Figure 6-3 shows the application of the algorithm on the PDE with non-constant coefficients. In this case, we observe an increase in accuracy in the same order of magnitude as seen earlier. However, in this case the gaps between the various partitions is not as distinct as seen in the previous two examples.
Figure 6-4 shows the gap in the value of $m_{0,0}$ of the heat equation. In this case we see much higher orders of accuracy. By using just two partitions, we reduce the gap to $10^{-8}$ even though this is only slightly better than the no-partition case. One should observe that the higher partition systems are more accurate for lower values of $N$. But as we increase the number of moments and partition the domain further, we run into numerical problems. Under such circumstances, SeDuMi stops solving the optimization problem with the gap between the primal and dual solutions higher than in the case of using no-partitions or two partitions. Hence, the problem seemed to be going in the correct direction but then ran into numerical problems. Therefore, the solutions are much worse in this case than when using no-partitions or only two partitions.

This phenomenon of running into numerical problems was observed for many other test cases as well. For a small $N$, the higher partition cases performed really well. But a combination of high $N$, and higher partitions resulted in numerical problems, which gave poor bounds. However, this was not an indication of failure of the algorithm. Instead, the problem should be solved taking into account any special structures of the matrices involved in the problem.

6.2.3 Reduction of feasible region

Another interesting observation was that in some problems for small $N$, the non-partitioned case calculated an extremely tight bound for a particular objective. However, when the number of test functions were increased, or the domain was partitioned, the solver reported that the problem was infeasible. When the analytical moments were calculated, and put into the linear and semidefinite constraints equations, they still satisfied all the equations. Hence, this suggests that the feasible region is shrinking to a small region, or even a point, which the solver is unable to find. This again indicates that the algorithm is reducing the feasible region of the problem, and thus, we should find tighter bounds when the problem is partitioned.
6.2.4 Relaxation of constraints

In the cases, where numerical errors are encountered, or the SDP solver cannot find a solution, we found that by not imposing all the constraints mentioned above, we can obtain better bounds on the solution. There are two ways of relaxing the constraints:

1. We can generate lesser number of linear equations for a given value of \( N \), but keep the semidefinite constraints matrices of the same size. For example, if \( N = 4 \), we will generate linear constraints using \( 1, x, y \ldots y^3 \) but will not generate the linear constraints using \( x^4, x^3y, x^2y^2, xy^3 \) and \( y^4 \). However, the semidefinite constraint matrices will include these monomials. We have found that many times allows us to obtain tighter bounds.

2. We also found that the tighter lower bounds of the objective functions can be obtained by not imposing the semidefinite constraints on the boundary moments \( (z_\alpha) \). Therefore, we can use higher degree test functions to obtain tighter lower bounds on the objective. However, if these semidefinite constraints are not imposed, upper bounds are not obtained, as SeDuMi reports the problem to be unbounded.

6.3 Summary

The key points to be taken from the examples presented are:

- By partitioning, we are able to get more accurate answers without using higher order test functions. The number of linear equations and size of the semidefinite constraints increases in \( O(N^2) \), as \( N \) is increased. Hence, it is quite beneficial to solve these systems using small values of \( N \).

- In some cases, for higher number of partitions, we ran into numerical problems. In other cases, the feasible region seems to be reducing to a small region; hence the SDP solver cannot solve the problem. This seems to indicate that the effectiveness of the partitioning is dependent on the nature of the solution.
• We can relax some of the constraints to solve problems if there are numerical issues or the feasible region becomes very small. This allows us to obtain tighter bounds for some problems.
Chapter 7

Conclusions and future directions

We have presented a modification to an existing methodology to calculate bounds on linear differential equations. The key steps in the algorithm are

- Partition the domain into smaller sections. The number of partitions is an input into the algorithm.
- Translate each partition to the domain $[-1, 1]$. Map the original domain to a normalized coordinate system to ensure numerical stability.
- Use the adjoint operator to write linear equations for each of the partition.
- Introduce coupling conditions at the boundary between adjacent partitions.
- Write semidefinite constraints for each of the partitions.
- Solve the system of equations to obtain moments for each of the partitions. Using these, recover the original moments.

The algorithm gives lower and upper bounds on the moments, using which the actual value can be inferred. For the one-dimensional ordinary differential equations, partitioning gave excellent benefits over the original algorithm. In some cases we increased the accuracy by nine decimal places. Partitioning did not seem to provide great benefits in the cases where the solution was a polynomial, as the non-partitioning method gave accurate results.
In the two-dimensional case, accuracy certainly improved after partitioning. Although the improvement was not as good as that observed in the one-dimensional case, it did seem to be promising. The greatest benefit was getting the same accuracy as the non-partitioned case and high degree of test function, by using lower degrees of test functions and partitioning the domain. In some cases, it seems that partitioning reduces the feasible region to a point, hence the solver was unable to find a solution. However, we did encounter some numerical stability issues.

Currently, the implementation is for linear differential equations with polynomial coefficients and monomials as test functions. This can be expanded to allow general coefficients and test functions. This can increase the scope of differential equations whose moments can be calculated, and maybe the stability of the solutions for certain families of solutions. Further investigation is required on when partitioning can reap benefits. In our work, we have loosely stated that if the function is complex or has variations, partitioning is good; this needs to be formalized.

We have currently implemented equal partitions in the domain. This can be modified, and partitions of different sizes depending on the problem solution can be introduced. For future development, a robust tool can also be developed, which will calculate the moments for a particular differential equation using the most efficient way, i.e. with or without partitioning. Such a tool might even allow the user to specify or choose between accuracy and speed of moment calculation. The hope is that in the future this method can complement the already existing PDE solvers.
Appendix A

Expansion of terms in the two-dimensional equations

The expansion for each of the terms in the two-dimensional case are provided below.

\[
\int u_{xx} \phi dx dy = \int \frac{\partial^2 u}{\partial x^2} \phi dx dy \\
= \int \phi \frac{\partial^2 u}{\partial x^2} dx dy \\
+ \int_{k_1}^{k_2} u_x(x = k_2, y) \phi(x = k_2, y) dy \\
- \int_{k_1}^{k_2} u_x(x = k_1, y) \phi(x = k_1, y) dy \\
- \int_{k_1}^{k_2} u(x = k_2, y) \phi_x(x = k_2, y) dy \\
+ \int_{k_1}^{k_2} u(x = k_1, y) \phi_x(x = k_1, y) dy
\]

91
\[ \int u_{yy} \phi dx dy = \int_{\Omega} \frac{\partial^2 u}{\partial y^2} \phi dx dy \]
\[ = \int_{\Omega} u \phi_{yy} dx dy \]
\[ + \int_{k_1}^{k_2} u_y(x, y = k_2) \phi(x, y = k_2) dx \]
\[ - \int_{k_1}^{k_2} u_y(x, y = k_1) \phi(x, y = k_1) dx \]
\[ - \int_{k_1}^{k_2} u(x, y = k_2) \phi_y(x, y = k_2) dx \]
\[ + \int_{k_1}^{k_2} u(x, y = k_1) \phi_y(x, y = k_1) dx \]

\[ \int u_{xy} \phi dx dy = \int_{\Omega} \frac{\partial^2 u}{\partial y^2} \phi dx dy \]
\[ = \int_{\Omega} u \phi_{xy} dx dy \]
\[ + \int_{k_1}^{k_2} u_y(x, y = k_2) \phi(x, y = k_2) dx \]
\[ - \int_{k_1}^{k_2} u_y(x, y = k_1) \phi(x, y = k_1) dx \]
\[ - \int_{k_1}^{k_2} u(x, y = k_2) \phi_x(x, y = k_2) dx \]
\[ + \int_{k_1}^{k_2} u(x, y = k_1) \phi_x(x, y = k_1) dx \]

\[ \int \frac{\partial u}{\partial x} \phi dx dy = - \int_{\Omega} \phi_x u dx dy + q_1 \int_{\partial\Omega} n_x u \phi dS \]
\[ = - \int_{\Omega} \phi_x u dx dy \]
\[ + \int_{k_1}^{k_2} u(x = k_2, y) \phi(x = k_2, y) dy \]
\[ - \int_{k_1}^{k_2} u(x = k_1, y) \phi(x = k_1, y) dy \]
\[
\int_{\Omega} \frac{\partial u}{\partial y} \phi dxdy = - \int_{\Omega} u \phi_y dxdy + q_1 \int_{\partial \Omega} n_y u \phi dS
\]

\[
= - \int_{\Omega} u \phi_y dxdy + \int_{k_1}^{k_2} u(x, y = k_2) \phi(x, y = k_2) dx
\]

\[
- \int_{k_1}^{k_2} u(x, y = k_1) \phi(x, y = k_1) dx
\]

\[
\int_{\Omega} u \phi dxdy = \int_{\Omega} u \phi dxdy
\]
Bibliography


