ON THE SOLUTION OF MIXED BOUNDARY VALUE PROBLEMS IN ELASTICITY

by

Michael Hohn

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mathematics

The University of Utah

December 2001

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Michael Hohn

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: E.S. Folias

Peter Alfeld

Nelson H.F. Beebe

Reaz Chaudhuri

Frank Stenger

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of ___________Michael Hohn___________ in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

_________________________ ____

Date

E.S. Folias
Chair, Supervisory Committee

Approved for the Major Department

____

James A. Carlson
Chair/Dean

Approved for the Graduate Council

____

David S. Chapman
Dean of The Graduate School

ABSTRACT

A method and algorithm for the solution of linear two-dimensional first-order systems of elliptic partial-differential equations (PDEs) and associated boundary conditions over a finite union of rectangles using sinc-collocation methods, collectively called sinc-ellpde, are presented.

An overview and short description of the stages of the method and steps of the algorithm are given along with a trivial sample problem. These are followed by detailed descriptions of two problems from fracture mechanics, the method, and the numerical results obtained. The algorithm is then described in detail, followed by a convergence proof, followed by full descriptions of the symbolic computations required for the fracture mechanics problems.

This work is dedicated to the computers and programs that made it possible, and pleasant, to produce: Unix systems and tools, and the Ocaml programming language.

CONTENTS

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to begin by thanking the faculty members who supported and mentored me throughout this research. The original fracture mechanics problems, which are the driving force behind this work, were suggested by E. S. Folias, who also provided funding for several years. The use of sinc-based methods for solution of these problems was suggested by Frank Stenger, and the method presented here builds on and extends results from his and other people's work over the last many years. The largest part of the work was programming related. Through countless discussions with Nelson H. F. Beebe, I gained much insight into broad areas of computer science, especially those of practical utility. Reaz Chaudhuri was of great help in obtaining a proper understanding of the solid mechanics equations. He always pointed me to proper references, and we had many interesting discussions. My master's thesis, done with Peter Alfeld, gave much preparation in the area of numerical linear algebra, especially the solution of large sparse linear systems. I also wish to extend my gratitude to Graeme Milton for several discussions, and providing funding during the period in which the contents of Appendices A and B were written.

On the personal side, thanks go to all my friends who provided support and encouragement, and helped me stay sane. Marzenka, Alex, Brent, Ross, Pieter, Andrew, Pete, I thank you all. Many thanks to Frank for having great parties with great food.

CHAPTER 1

INTRODUCTION

1.1 Motivation

In design, anticipation of material failure is vital. Over the last decades, a theory first proposed by Griffith ((Griffith 1924)) has become one of the most-used tools in prediction of fracture of simple materials. In the Griffith theory, a material is assumed to have microscopic cracks with high stresses found at their tips; these cracks cause the material to fail at a much lower level of force than molecular binding forces predict. More recently introduced composite materials also have cracks, and the governing equations are more involved because of material interactions.

To accurately predict failure of these materials requires knowledge of the stress field near the crack tips as well as the rest of the material. For all but the simplest geometries, closed-form solutions are almost impossible to obtain, so numerical methods have become very popular. For engineering mechanics, by far the most popular methods are the finite element (FE) and boundary element methods (BEM). Some of their desirable properties are relative conceptual simplicity, straightforward (but tedious) implementation, straightforward handling of complex geometries, a large available code base for solving problems, and a tendency to require only modest computing resources. For crack problems and other problems with corner or edge singularities it is common to use special elements to handle these singularities, since plain FE cannot. Thus, a closed-form solution revealing the nature of the singularities must be available before a complete solution is obtained with FE or BE methods.

For two-dimensional crack problems, e.g., (Erdogan 1963) and (England 1965), the stresses at a distance r from the crack tips are shown to be proportional to $1/\sqrt{r}$, and the constant of proportionality, K_I, depends only on the geometry of the material. For some special three-dimensional crack problems, e.g., the penny-shaped crack described in (Sneddon 1946) and (Sneddon 1995, p. 427), stress fields are also found to be proportional to $1/\sqrt{r}$. This is enough information to complement the (polynomial based) FE methods with singular elements (which behave like $1/\sqrt{r}$ in the appropriate regions), thus reducing the problem to one for which FE are well suited, and resulting in good numerical answers.

For other two- and three-dimensional problems, however, asymptotic expansions have found stress fields at a distance ρ from the crack tip to be proportional to $\rho^{-\alpha}$, α depending on the geometry and parameters of the material. In (Folias 1975), the stresses near a corner of a cracked plate of finite height are shown to be proportional to $\rho^{-1/2-2\nu}$, where ν is Poisson's ratio. This allows a wide range of α; other results with $\alpha \neq 1/2$ are found in (Hein and Erdogan 1971; Bogy 1968).

Thus, one does not in general know the behavior of the singularity a priori, and the problem cannot be reduced to one readily handled by finite elements. Further, the closed-form solutions, e.g., (Folias 1975) and (Folias 1965), are not easily obtained; some recent solutions for more difficult problems ((Zhong and Folias 1992; Penado and Folias 1989)) are even more difficult to derive and are seminumerical, not truly closed-form. And for most three-dimensional problems, closed-form or seminumerical solutions are not yet available and may never be found.

It is natural, then, to change the emphasis from “patching” of existing numerical methods or development of seminumerical methods to developing new numerical methods with the ability to handle singularities without explicit knowledge of their behavior.

1.2 Sinc methods

The group of sinc-function based methods, introduced, e.g., in (Lund and Bowers 1992), are very promising here; they handle singular functions[1] and enjoy an exponential convergence rate ($2N+1$-term error proportional to $\sqrt{N}e^{-\sqrt{\pi d \alpha N}}$) enabling one to get many digits of accuracy with reasonable work. For crack and other singular problems, this means only the *location*, but not the type, of the singularity is needed, and the solution can be accurately computed.

There has been significant theoretical development of the sinc methods for one-dimensional problems in the areas of differential equations, integral equations, Hilbert transforms, definite and indefinite integration, and convolutions. In particular, the theoretical foundations justifying sinc interpolation and collocation on one domain are found in (Stenger 1993a), while several one-dimensional and some simpler two-dimensional examples are shown in mathematical detail in (Lund and Bowers 1992). Work on one-dimensional multiple domain problems using sinc methods has been done in (Morlet et al. 1997; Morlet, Lybeck, and Bowers 1999; Lybeck and Bowers 1996; Lybeck and Bowers 1994). On the practical side, (Stenger 1993b) provides a collection of FORTRAN routines for some of these one-dimensional calculus operations including quadrature, indefinite integration, differentiation, indefinite convolutions and Laplace transforms; descriptions for these are provided in (Stenger 1993c).

Unfortunately, implementation of two- or higher-dimensional sinc algorithms is much more difficult than their one-dimensional counterparts, and to date, only relatively simple higher-dimensional problems have been solved, those reducible to one dimension and those with trivial boundary conditions.

For reasons of implementation (see Section 1.3), the class of problems considered here are two-dimensional, multiple-unknown linear elliptic first-order system of partial differential equations and their boundary conditions, with or without corner and/or edge singularities, defined on a finite, connected union

[1] Given a finite function f on an interval $[a, b]$, near a only $|f(x) - f(a)| < C|x-a|^{\alpha}$ is required, $\alpha > 0$. A similar bound must hold near b.

of rectangles. Therefore, the method presented can be used for the solution of a whole class of problems, including the two-dimensional fracture problems mentioned in the beginning of this chapter.

The notation in the mentioned literature is well suited for mathematical proofs, but not at all suited for preparing a (computer) algorithm for a general class of problems, especially when working in two or more dimensions. Therefore, Chapter 4 concisely summarizes previous results and directly extends them to two dimensions, without proof; Chapter 6 provides detailed descriptions of the resulting structures and algorithm, and finally Chapter 7 details the two-dimensional definitions and proofs for the algorithm.

1.3 Programming considerations

The implementation of one-dimensional, single-unknown, single-domain sinc algorithms is relatively easy and can be done equally well (or poorly) in C, FORTRAN, Maple, Matlab, Octave, etc., in less than one week. This includes all core functionality, input/output, and visualization. The design and implementation of the present algorithm, in a high-level functional language, has taken just under 18 months, also including all core functionality, input/output routines, and visualization. Although the *method* is seemingly easy to extend from one to multiple dimensions, the *algorithmic implementation* of this extended version is thus vastly more complex.

There are several sources of this complexity. The first and most obvious is the generality of the method — why not just write a specialized program for the problem at hand? Simply put, this *will not work*. Even the shortest possible program for a real problem is substantial; there will be errors in the implementation of the method, independent of the programming language[2] used; finding these errors requires simple test runs with known intermediate results. Thus, the

[2] The first program written failed to produce useful results, and the errors were never found; this was due to an imperative coding style coming from FORTRAN, although programming was done in C.

algorithm must be able to handle many different problems.[3]

Handling of input/output data is a second source of complexity. When a program is partitioned into separate parts, there is suddenly a need to pass data between these pieces; the more partitioned the program, the more complex these data will be. The common impulse to add more features to an existing program (as opposed to adding more modules) can be attributed to the difficulty usually encountered in the exportation and importation of complex data. Therefore, data serialization is considered by the author to be one of the critical facilities a modern programming language *must* provide. Also critical is the reading of user-provided data. A complex problem will have user input data at least as complex. For the present algorithm, at a minimum the equations, boundary conditions, geometry information and problem parameters have to be provided. However, this is not the only use of these data, which are also used to provide a user-verifiable presentation of the problem it represents (in the form of graphics, equations, etc.). It is therefore critical to read and write the data correctly; this amounts to no less work than a typical compiler front end: lexer, parser and tree walker are all required.

A third problem is the method-specific visualization of computed data. There are many "general purpose" data-visualization environments available. Unfortunately, the ones found by the author are not suited to handle the peculiarities of the sinc method, and are severely limited in their programmability. So the only choice is to use low-level graphical and user-interface libraries, and create a custom environment.

These complications taken together really make the distinction between a simple self-contained implementation of an algorithm with no connection to the world and a full environment for solving a particular type of problem.

[3] A second program, written in object-oriented C++, worked on many test problems but as implemented, it required an excessive amount of user input making it impractical.

Lastly, the method implemented also had one defect: the solution had to have bounded derivatives, which excludes crack problems.

Thus, this implementation was hard to use and unsuitable for the original problem, and had to be rewritten.

In handling these complex programming problems, the *most important* factors influencing productivity are the *choice of programming language(s)*, choice of programming style, and availability of libraries with a clean interface.[4] Consider this 22-year-old quote from (Backus 1978):

> ... Discussions about programming languages often resemble medieval debates about the number of angels that can dance on the head of a pin instead of exciting contests between fundamentally differing concepts.

The same could be said today about arguments for or against the use of FORTRAN vs. C vs. JAVA vs. C++. While the latter three have significant advantages over FORTRAN, these languages are all fundamentally based on state modification, two of them with a nice object-oriented veneer.

Even though there is no single all-encompassing solution to programming problems, for the present work, good integration, in Ocaml, of the following language facilities and properties was *indispensable*:

- functional programming support (used almost exclusively)
- imperative programming support (used very rarely)
- expression-based language
- interactive toplevel
- garbage collection
- polymorphism
- complex data structure support – including pattern matching
- exceptions
- strong static typing

[4] There are many libraries with interfaces so poorly designed as to make the library unusable.

- type inference
- module support
- data serialization
- C language interface
- excellent debugging facilities
- efficient generated code

The necessity of these facilities could be argued, but there is no doubt that they lead to faster program creation, easier maintenance, fewer bugs and clearer structure. Programming in a language missing any one of these facilities is little better than programming in assembly language. The support for these facilities in some common languages is summarized in Table 1.1. In the table, the execution speeds are relative to C, as measured by some simple benchmarks pertinent to the present problem. A feature directly available is marked with a solid bar; the length indicates the ease of use or quality of the feature. A feature that is available through some additional tool is marked with a circle; using such a feature is usually not as straightforward as using a built-in equivalent. It should be noted that usually absence of a facility is fatal; there is no practical way to add it. Also, the presence of a feature, e.g., a C interface in JAVA, does not imply that its use is simple.

[5] C toplevels are slow, importing compiled code is tedious, and they have limited functionality.

[6] C polymorphism via typecasts is unsafe and the source of many errors

[7] C++ template polymorphism is *slow* to compile, produces large amounts of code, and any errors propagate after macro expansion making debugging a nightmare. Templates are a very weak macro mechanism, and are often (ab)used as full code generators; anything done via templates could be done more cleanly via use of, e.g., the m4 macro processor. Further, more powerful uses of macros are very difficult in statement-based languages, while in expression-based languages, very impressive extensions of the language are possible; see (Graham 1994).

[8] Exceptions are problematic in languages without garbage collection. In C++, only proper deallocation of the stack contents are guaranteed.

Table 1.1. Programming language features.

language	FORTRAN 77	C/C++	Java	Common Lisp	Ocaml
Programming Styles					
imperative	▬▬▬▬	▬▬▬▬	▬▬	▬▬▬▬	▬▬▬▬
Object Oriented	—	○/▬▬▬	▬▬▬	▬▬▬▬	▬▬▬
Functional	—	—	—	▬▬▬▬	▬▬▬▬
Language Features					
interactive toplevel	—	○[5]	○	▬▬▬▬	▬▬▬▬
garbage collection	—	○	▬▬▬▬	▬▬▬▬	▬▬▬▬
polymorphism	—	○[6] / ○[7]	○	▬▬▬▬	▬▬▬▬
pattern matching	—	—	—	○	▬▬▬▬
exceptions	—	○/ ▬▬[8]	▬▬▬▬	▬▬▬▬	▬▬▬▬
typing	weak	weak static	static/ dynamic	dynamic	strong static
static type checking	▬	▬▬	▬▬	▬▬	▬▬▬▬
type inference	—	—	—	N/A	▬▬▬▬
anonymous data structures	—	—	—	▬▬▬▬	▬▬▬▬
complex static data structures	—	▬▬▬▬	▬▬	▬▬▬▬	▬▬▬▬
complex dynamic data structures	—	▬▬	▬▬	▬▬▬▬	▬▬▬▬
modular programming	—	○/▬▬	▬▬▬▬	▬▬▬	▬▬▬▬
data serialization	—	—	▬▬▬▬	▬▬▬	▬▬▬▬
Other					
language style	statement	statement	statement	expression	expression
standard library	—	poor/good	excellent	excellent	excellent
debugging facilities	good	good	vary	vary	good
C interface	▬▬▬▬	▬▬▬▬	▬▬	○	▬▬▬
execution speed	1.0	1.0	2.0 - 50.0	1.0 - 10.0	1.0 / 17.0

The choice of programming language must be appropriate for the problem at hand; as can be seen from the table, FORTRAN, C and C++ are *not* suitable languages for the present project. Default choices are usually bad, but seem to recur without end. In scientific computing, the use of FORTRAN is pervasive; it is the de facto standard. This is ironic, as FORTRAN's inventor heavily criticizes the imperative (or von Neumann) style of programming supported by the FORTRAN 77 dialect (Backus 1978). Unfortunately, the author's first attempt of an algorithm implementation was done in this style — and failed.

The new features introduced in FORTRAN 90, etc., do not replace the existing computational model; they merely add some features already found in other languages, e.g., C++: structured programming support, data structures, free-form input, etc. Recently, more numerical code has been written in object-oriented C++, under the premises of better readability and reusability. For the present algorithm, a second implementation was written this way. Although it worked for many test problems, there were severe shortcomings, all due to a poor choice of language, C++[9].

1.4 Summary

One purpose of the present work is the extension of sinc methods to solve a certain class of two-dimensional elliptic boundary value problems, including several encountered in fracture mechanics. Development of method and algorithm has been guided by two representative problems. The first, a single material with a crack along one axis, has been solved by (Sneddon 1946) and others for the case of an infinite plate, under plane-strain or plane-stress assumptions. The second, a bimaterial plate with a crack along the interface, was solved by (England 1965) and (Rice and Sih 1965). These problems are described in Sections 3.1 and 3.2, respectively.

A second purpose of this work is the presentation of a general method for solving a whole class of two-dimensional, multiple-unknown, linear elliptic first-order systems of partial differential equations and their boundary conditions, with or without corner and/or edge singularities, over a union of rectangles. In fact, to the algorithm the only difference between the mentioned problems is the number of rectangles. The sinc method of solution is described in Chapter 4, and numerical results are presented and compared to known closed-form solutions in Sections 5.1 and 5.2.

[9] The program was nearly unreadable, due to unnecessary clutter typical of C++ and other statically typed, rigid object-oriented languages: type and class declarations, explicit memory handling, poor exception handling support, excessive data hiding resulting in many (superfluous) accessor functions, and the underlying idea of state-based programming.

Some of these problems would not occur in dynamically-typed object-oriented programming languages; unfortunately, dynamic typing introduces another source of problems.

As may be expected, this algorithm is complex; it should be no surprise that its implementation as a computer program is even more complex. Therefore, the third purpose of this work is to illustrate and make a strong case for the use of modern tools from computer science in the design and practical algorithmic implementation of a complex numerical algorithm. The importance of this third aspect cannot be overstated. After an initial failure to produce a working monolithic implementation of the predecessor of the present algorithm, followed by a working but unreadable, nonextensible, nonmaintainable implementation of the algorithm summarized in Appendix C, the importance of using proper programming languages and techniques became painfully apparent.

CHAPTER 2

A COMPLETE EXAMPLE

The elasticity problems, mathematics, and algorithms to be described are each complex enough to fill several chapters. It would be unreasonable to expect a reader to understand all those details and from them reconstruct an overall picture of this work. Therefore, this chapter presents a simpler problem, a very short overview of the SINC-ELLPDE algorithm, and a complete walk-through of the steps needed to obtain a set of solutions. Using these solutions and the exact answer, the accuracy of approximated function *and* derivative values are measured, and the convergence rates are compared to their theoretical bounds.

2.1 Problem formulation

This problem considered here is a single-rectangle version of sample problem 5 from the TTGU manual (Kaufman 1990): Laplace's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

on the rectangle $(0,1) \times (0,1)$, with Dirichlet conditions on left, top, and right boundaries, and Neumann conditions on the bottom boundary.

For the present problem, the exact solution is taken to be the real part of $z \ln(z)$ or

$$f(x,y) = \frac{1}{2}\, x \ln(x^2 + y^2) - y \arctan(y,\, x),$$

providing a weakly singular solution and an excellent test for the SINC-ELLPDE method. For this solution, the partials in x and y are

$$f_x(x,y) = \frac{1}{2} \ln(x^2 + y^2) + 1$$

and

$$f_y(x, y) = -\arctan(y, x),$$

respectively.

2.2 Method of solution

The method of solution used here is based on collocation of the partial differential equation (PDE) and the boundary conditions (BCs) on a finite union of rectangles. The current algorithm is specialized for two-dimensional linear elliptic systems and proceeds as follows.

Block conversion: For every rectangle, both the PDE and the BCs are written as a collection of first-order systems; in this collection, every unknown is replaced by a sinc series of the form

$$\sum_{i=-N_1}^{N_1} \sum_{j=-N_2}^{N_2} c_{ij}\omega_i(x)\omega_j(y); \tag{2.1}$$

and the corresponding differential operator is applied to this new form.

Discretization: For every rectangle, the resulting collection of systems is then discretized via evaluation of these series at the sinc collocation points

$$z_{ij} = (\psi(ih_1), \psi(jh_2)). \tag{2.2}$$

The discretizations from all rectangles are then combined into one linear system

$$[L][u] = [b] \tag{2.3}$$

Solution: This linear system is large and sparse, and is solved using the SuperLU package presented in (Demmel, Eisenstat, Gilbert, Li, and Liu 1999).

Reconstruction: The sets of coefficients c_{ij}, for all unknowns on all rectangles, are then extracted from the resulting solution vector $[u]$ and every original unknown is approximated using a series of the form of Equation 2.1.

Given an unknown f and its sinc series approximation $\overline{f}$ from Equation 2.1, the bound for the absolute error is given by

$$\epsilon_N \ = c\sqrt{N}\exp(-g\sqrt{N}) \tag{2.4}$$

for the function, and by

$$\partial\epsilon_N \ = cN\exp(-g\sqrt{N}) \tag{2.5}$$

for first derivatives.

Most of the details of the method of solution are handled automatically by the algorithm implementation. Usually, only the input equations, sinc and geometry data, and an evaluation point grid need to be provided. For the present problem, the exact solution is also needed to provide the boundary conditions' values, and for convergence checks.

2.2.1 Block conversion

The first-order system form is easily obtained. For consistency, define $u_1 \equiv u$. By defining the additional unknowns u_{11} and u_{12} as

$$u_{11} \quad = \partial u_1/\partial x \tag{2.6}$$

$$u_{12} \quad = \partial u_1/\partial y, \tag{2.7}$$

the second-order equation becomes

$$\partial u_{11}/\partial x + \partial u_{12}/\partial y \ = 0 \tag{2.8}$$

The two definitions and this equation form the set of interior equations for domain 1, the only domain (rectangle) for this problem. The top, left, and right boundaries use the simple Dirichlet condition $u_1 = f(x,y)$; using this condition along with Equations 2.6 and 2.7 gives the set of equations for the top, left, and right boundaries of domain 1.

The input equations must be provided in a very specific format; for this problem, they take the form

```
equation_specs=[
   unknowns=[u1, u11, u12],
   domains=[
      1=[
         regions=[
            Interior=[
               u11\ 1+~ u12\ 2=~0, u11-~ u1\ 1=~0,
               u12-~ u1\ 2=~0],
            Top=[
               u1=~f(x, y), u11-~ u1\ 1=~0, u12-~ u1\ 2=~0],
            Left=[
               u1=~f(x, y), u11-~ u1\ 1=~0, u12-~ u1\ 2=~0],
            Right=[
               u1=~f(x, y), u11-~ u1\ 1=~0, u12-~ u1\ 2=~0],
            Bottom=[
               u1\ 2=~0, u11-~ u1\ 1=~0, u12-~ u1\ 2=~0]]]]]
```

The program also needs the exact solution f for the boundary condition. This is provided as

```
let exact__ x y =
   let ln = log in
   let arctan = atan2 in
   (((1.0/. 2.0)*. x)*. ( ln  (x** 2.0+. y** 2.0) )-.
    y*. ( arctan  y  x ));;
```

2.2.2 Discretization

To convert the systems of equations to discrete systems, the geometry and sinc parameters are also needed by the program. An example for $N = 12$ follows.

```
(* Input data. *)
let pi = 3.1415926535897932385;;
let po2 = pi/.2.0;;
let dom_num_grid = [|(* Domain numbers, starting from 1. *)
  [| 1 |] ;
|]
and dom_x_bounds =     (* Positions of the boundaries.*)
  [| 0.0 ; 1.0 |]
and dom_y_bounds = [|
  1.0 ;
  0.0 ;
|]
and dom_x_sinc_alpha = (* Sinc approximation parameters. *)
```

```
  [| 1.0  |]            (* Horizontal stacking in x. *)
and dom_y_sinc_alpha = [|
  1.0 ;                 (* Vertical stacking in y. *)
|]
and dom_x_sinc_d =
  [| po2 *. 0.99 |]
and dom_y_sinc_d = [|
  po2 *. 0.99;
|]
and dom_x_terms =       (* Series summation limits. *)
  [| (12, 12); |]
and dom_y_terms = [|
  (12, 12) ;
|]
and gTrue_unknowns = ["u1"] and
    g1st_order_sys_unknowns = ["u11"; "u12"]
;;
```

2.2.3 Solution

Solution of the resulting linear system is automatic, so no input is needed here.

2.2.4 Reconstruction and evaluation

The reconstruction of the coefficients c_{ij} is automatic, but to be able to compare a sequence of solutions, the unknowns to be evaluated must all be evaluated at the same points. The choice of unknowns is made via the nested list

```
functions_to_plot=
       [    [domain, name, operator]=[1, u1, I],
            [domain, name, operator]=[1, u11, I],
            [domain, name, operator]=[1, u12, I],
            [domain, name, operator]=[1, u1, D2],
            [domain, name, operator]=[1, u1, D1]]
```

A nonuniform regular grid placing emphasis near the boundaries is the natural choice for singular problems as well as sinc methods; the grid is specified in the following form, shortened here for display:

```
full_info=[
   1=[
      x1=[
         num_pts=57,
```

```
    pts=[
       1.5998974884153694e-03, ...
       9.9840010251154132e-01],
    sinc_bounds=[3.9907906932903902e-05,
                 9.9996009209306713e-01],
    geometry_bounds=[0.0000000000000000e+00,
                     1.0000000000000000e+00]],
 x2=[
    num_pts=57,
    pts=[
       1.5998974884153694e-03, ...
       9.9840010251154132e-01],
    sinc_bounds=[3.9907906932903902e-05,
                 9.9996009209306713e-01],
    geometry_bounds=[0.0000000000000000e+00,
                     1.0000000000000000e+00]]]]
```

This point grid is shown graphically in Figure 2.1.

2.3 Numerical results

To obtain the data for illustration of the sinc convergence rate and general convergence behavior, the discretization, solution, and reconstruction steps are run several times, each time varying only N.

In the display of these data, the following are needed:

- a simple global view of convergence or lack thereof;

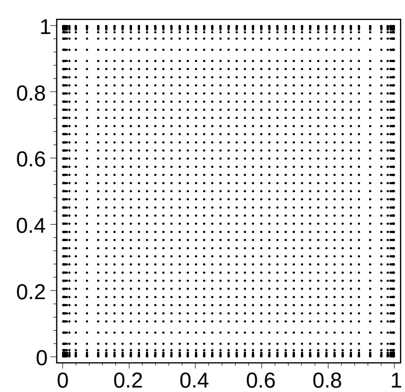


Figure 2.1. Grid for function evaluations. This grid is 57×57, with most points focused near the boundaries.

- a good qualitative idea of the global behavior of the function approximation;
- a precise local view of the function approximation and its quality.

To meet these requirements when examining singular problems or problems with boundary layers,[1] the solution and convergence are shown in two ways. The first shows the L^1 norm of the absolute error vs. N — a standard approach. In the second, the pointwise convergence behavior is shown using a collection of slices in the x and y direction, i.e., graphs of $f(x, y_j)$ vs. x for several fixed y_j and graphs of $f(x_i, y)$ vs. y for several fixed x_i. For identification of these slices and understanding the function as a whole, these are accompanied by a three-dimensional graph of the surface $\overline{f}(x, y)$

2.3.1 Convergence in norm

The error bounds of Equations 2.4 and 2.5 are sharper for large N. As the convergence is exponential, the vertical scale on a regular xy-graph would be dominated by the large errors occurring for smaller N; to avoid these difficulties, the logarithms of the error bound, $e_f(N)$, is fitted to the logarithm of the absolute error, $\ln|f - \overline{f}|$. Using a logarithmic scale for Equation 2.4, the error bound for function approximation becomes

$$e_f(N) \; = (\log(c) + \log(N)/2 - g\sqrt{N})/\log(10) \tag{2.9}$$

while the derivative error, from Equation 2.5, is bounded by

$$e_{\partial f}(N) \; = (\log(c) + \log(N) - g\sqrt{N})/\log(10). \tag{2.10}$$

Similar problems arise in the display of the computed and theoretical error. On a graph, a logarithmic vertical scale gives a much more practical picture, as

[1] For singular problems, or problems with boundary layers, the maximum norm would show very large absolute errors when in fact only small regions have large errors. The p-norms of the absolute error, $1 \leq p < \infty$, all weigh the error by area, avoiding this problem. But normwise convergence checks give only a global indication of convergence and say nothing about the local quality of approximation — unless the maximum norm is used.

the line remains almost straight and equal vertical space is used at all error measurements. Additionally, a graph of $\log_{10}(|f - \overline{f}|)$ vs. N, in which the vertical axis shows the number of accurate digits as linearly incrementing integers, is easier to view than a graph with logarithmic vertical axis which shows the error as floating-point number increasing by decades.

Figures 2.2, 2.3, and 2.4 show the convergence of u, u_x and u_y, respectively, using the $\log_{10}(|f - \overline{f}|)$ vs. N approach. To avoid the mentioned low-N inaccuracies, the points $N < 15$ were purposely ignored in the curve fits of the theoretical error bounds.

The figures show excellent agreement between the theoretical- and computed errors for $N \geq 15$, confirming the exponential convergence rate. Further, the theoretical value for g is given by $g = \sqrt{\pi d \alpha}$; with the default choices $d = \pi/2$ and $\alpha = 1$, $g \approx 2.22$, which is close to the computed values of 1.89105, 2.2833, and 2.19852, respectively.

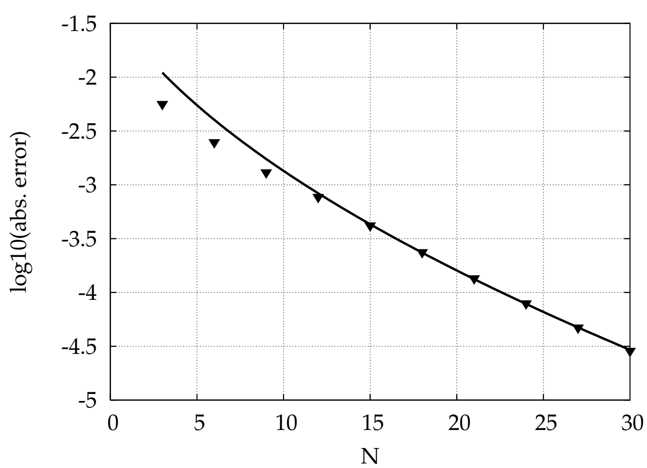


Figure 2.2. Absolute error as function of N for u. The error is in the L^1 sense. The curve is given by $c\sqrt{N}\exp(-g\sqrt{N})$, with $c = 0.168476$ and $g = 1.89105$.

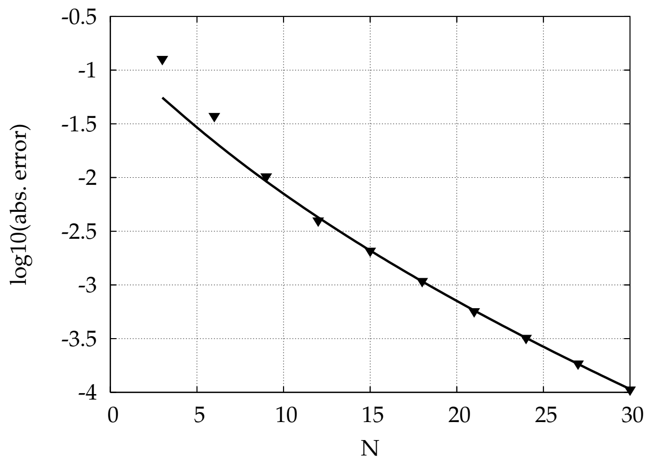


Figure 2.3. Absolute error as function of N for u_x. The error is in the L^1 sense. The curve is given by $cN\exp(-g\sqrt{N})$, with $c = 0.965548$ and $g = 2.2833$.

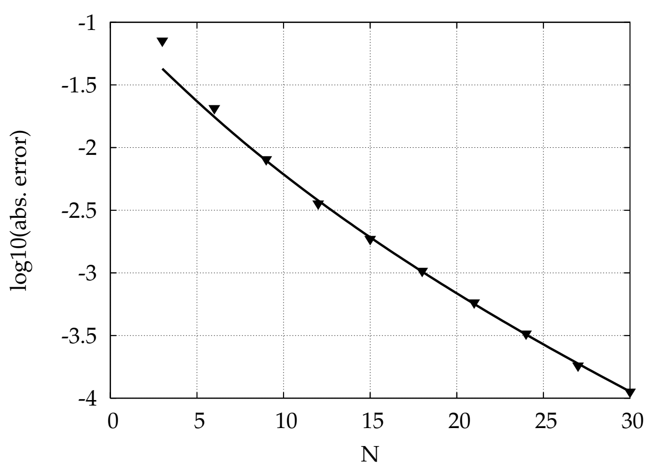


Figure 2.4. Absolute error as function of N for u_y. The error is in the L^1 sense. The curve is given by $cN\exp(-g\sqrt{N})$, with $c = 0.639411$ and $g = 2.19852$.

2.3.2 Pointwise convergence

To provide some insight into the pointwise convergence behavior of the sinc method over different areas of a given rectangle, the graphs in Figures 2.5 – 2.10 show a combination of three-dimensional surface- and two-dimensional xy-plots. The surfaces provide both a qualitative overview and a reference frame, and the xy-plots provide quantitative data along slices.

The graphs for every unknown are partitioned into several sets of figures. The first set of figures consist of (1) the "Area location" graph, displaying the current data's location via a solid rectangle and the computational rectangle via solid lines, the (2) "slice legend" table displaying the legends for the xy-slices, and (3) the "Area-surface view" graph, displaying a surface view of the data. The surface view shows the surface formed by the data; lines on the surface and their projections onto the base show the location of the xy-slices. The base projections are numbered for cross-reference with the xy-slices' graphs.

The second set of figures (and others, if present) shows the detail slices' xy-graphs. Each slice is numbered according to its position on the area-surface-view graph.

Some observations can be made here. Comparing Figures 2.6 and 2.8, the approximation for u converges more quickly than those for u_x, as expected from the error bound. More interesting is the behavior of the derivative approximations near the boundaries of the rectangle. As can be seen in Figure 2.8, the approximation is very good in the interior of the rectangle, but noticeably worse near the boundaries. For $m = 7$, this near-boundary error is quite severe; for $m = 19$, it seems small on the range used here (all of the rectangle). To better illustrate this characteristic, Figure 2.9 provides a closer look at a small area near $(0, 0)$, using larger values of N. In Figure 2.10, it is seen that for $m = 19$, good accuracy is obtained to about $x = 0.03, y = 0.03$, but the accuracy quickly diminishes when moving closer to the boundary. Increasing the number of terms from $m = 19$ to $m = 38$ again restores accuracy, showing the same drastic change of accuracy near the boundary observed on the full rectangle (Figure 2.7) for

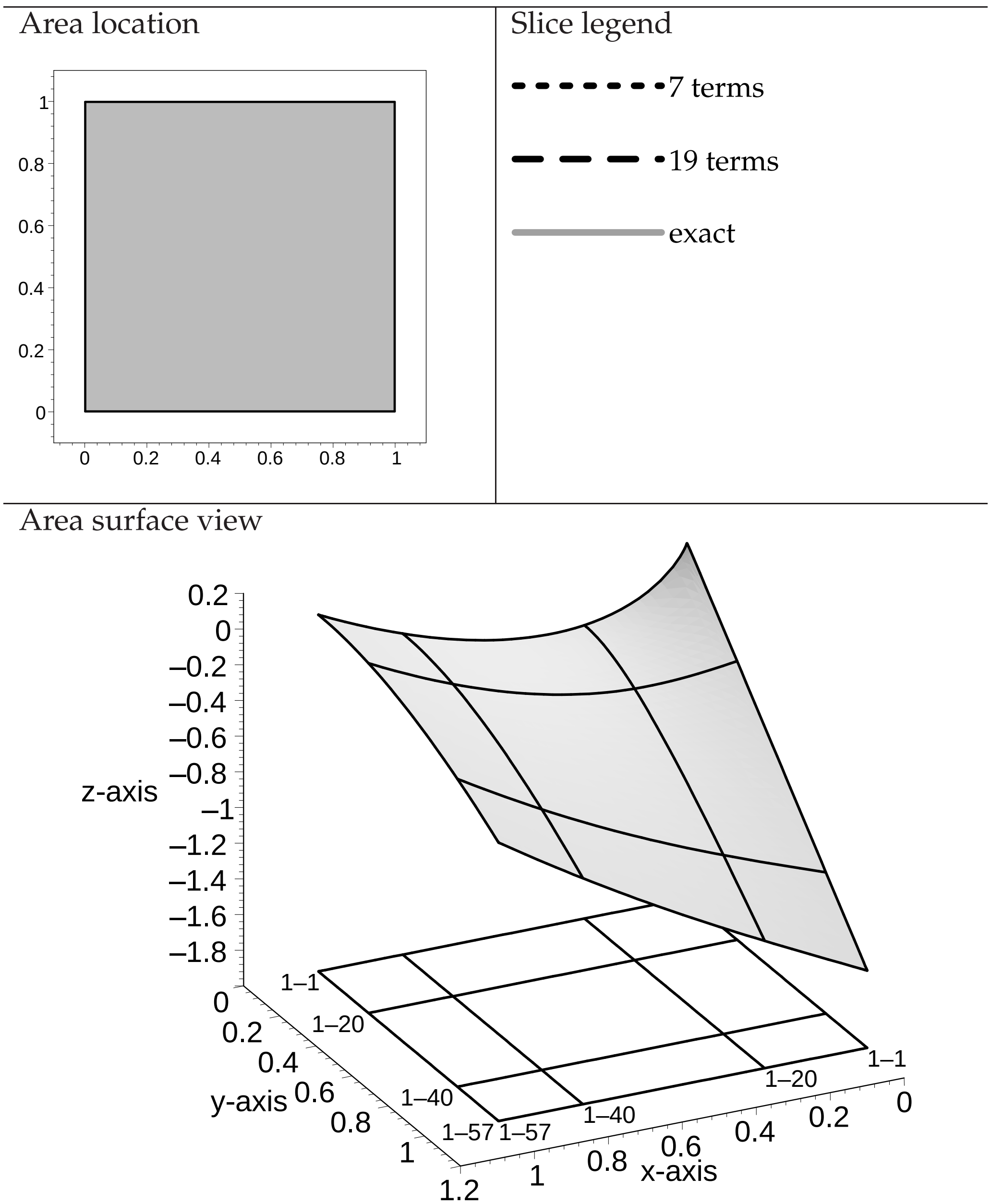


Figure 2.5. Graph displaying u as surface, the locations of slices, and the legends of the detailed pointwise error graphs shown in Figure 2.6.

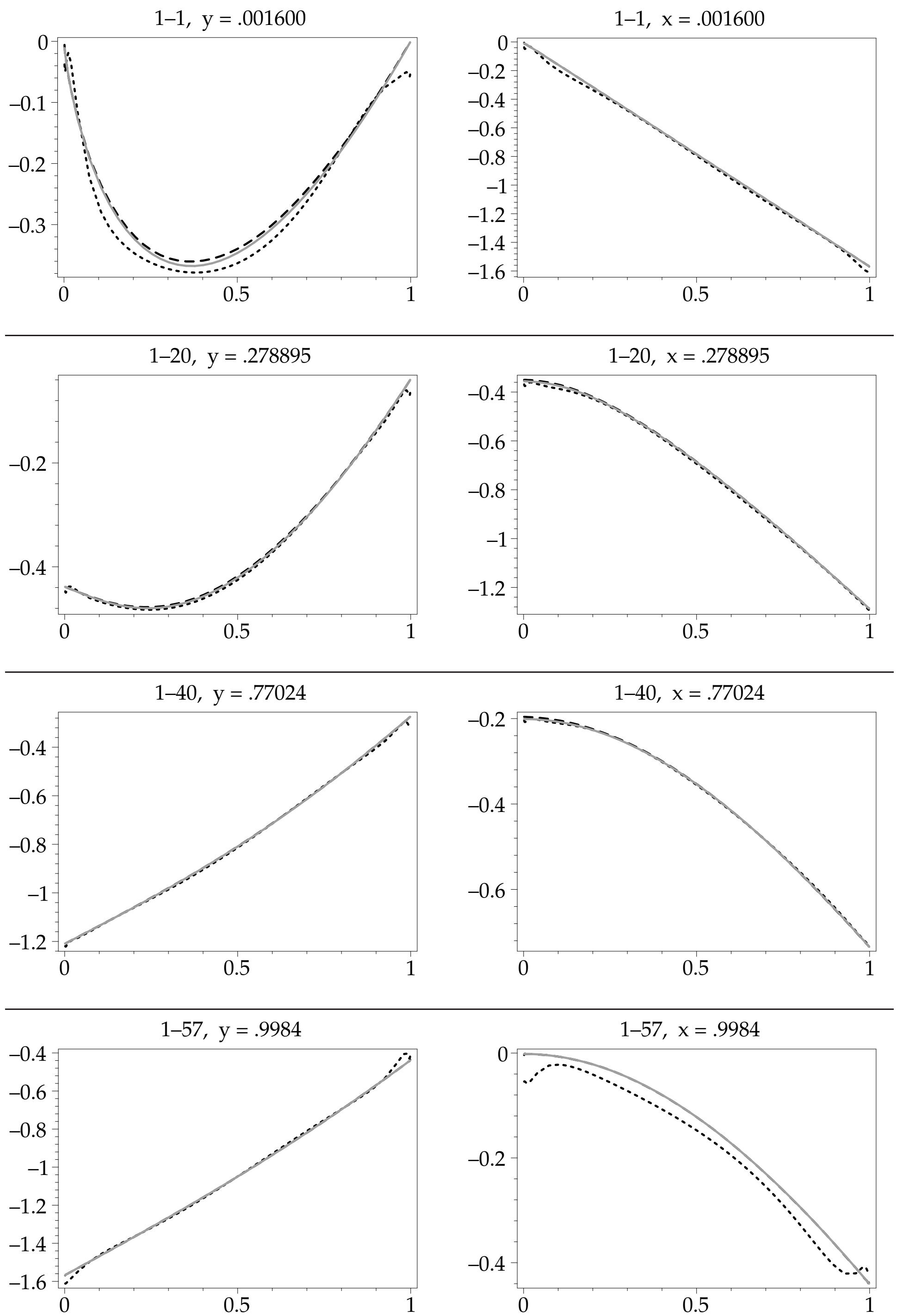


Figure 2.6. Graphs of slices detailing the pointwise error in the approximation of u. The locations of these slices are shown, by index, in Figure 2.5.

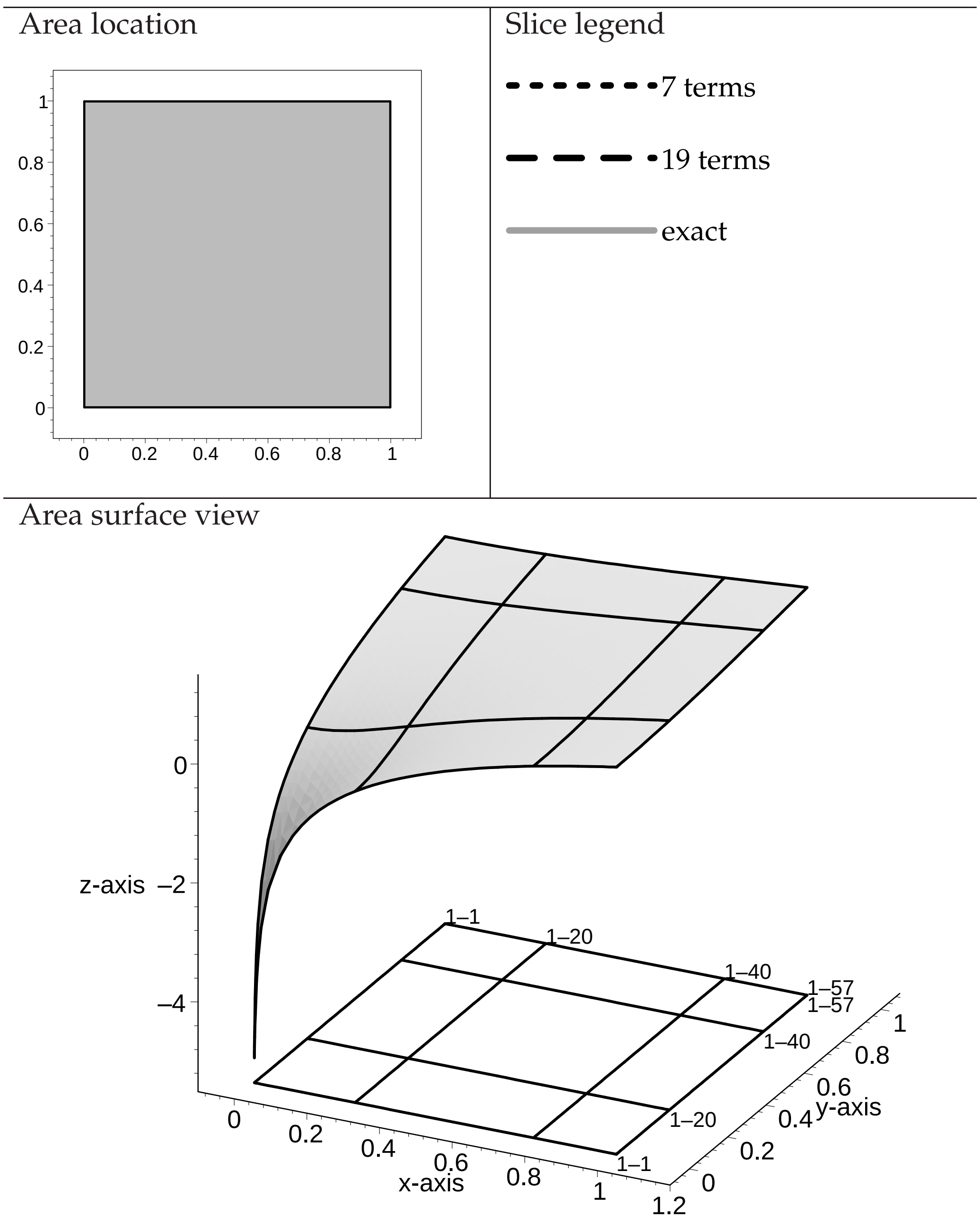


Figure 2.7. Graph displaying u_x as surface, the locations of slices, and the legends of the detailed pointwise error graphs shown in Figure 2.8.

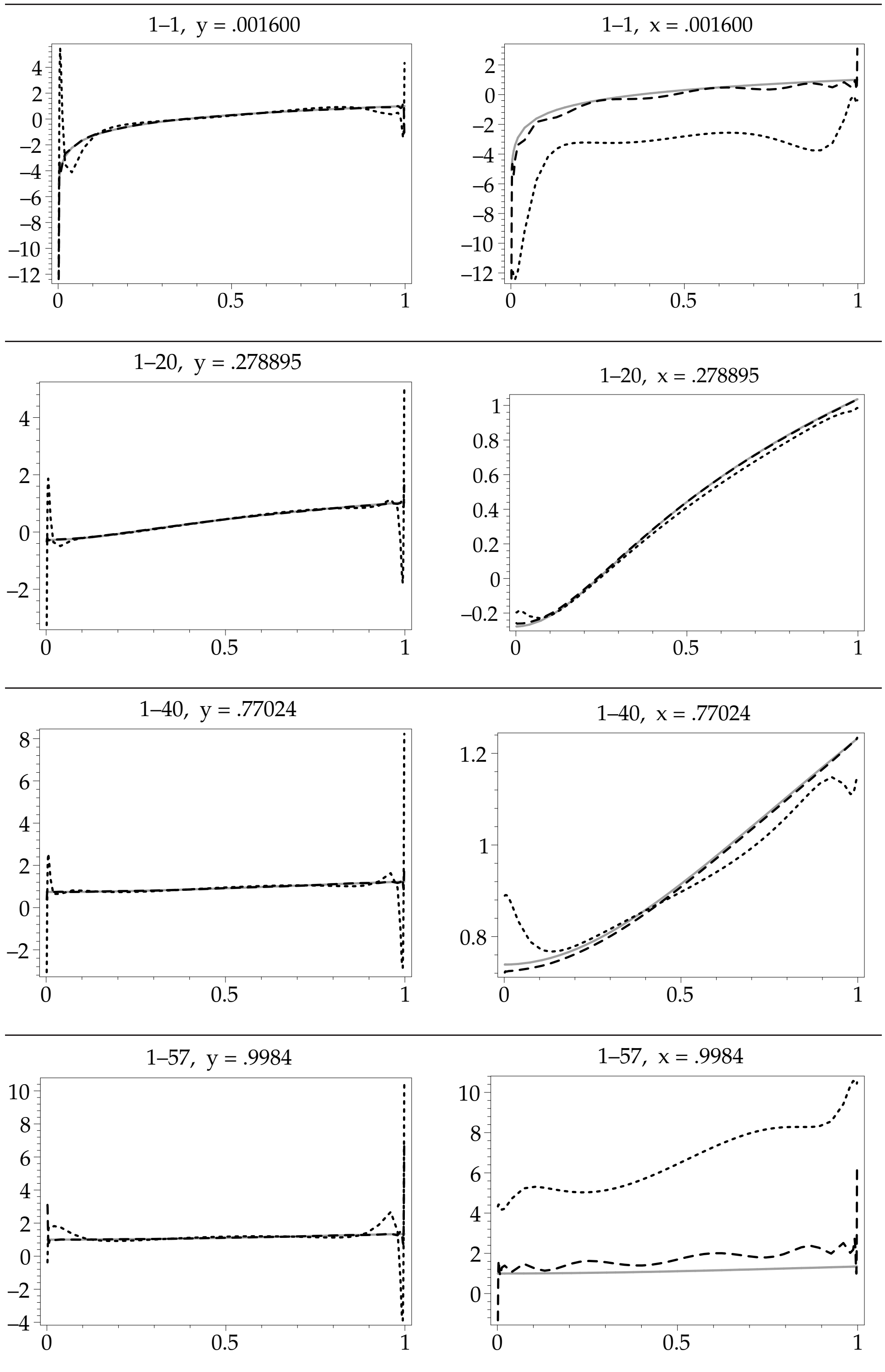


Figure 2.8. Graphs of slices detailing the pointwise error in the approximation of u_x. The locations of these slices are shown, by index, in Figure 2.7.

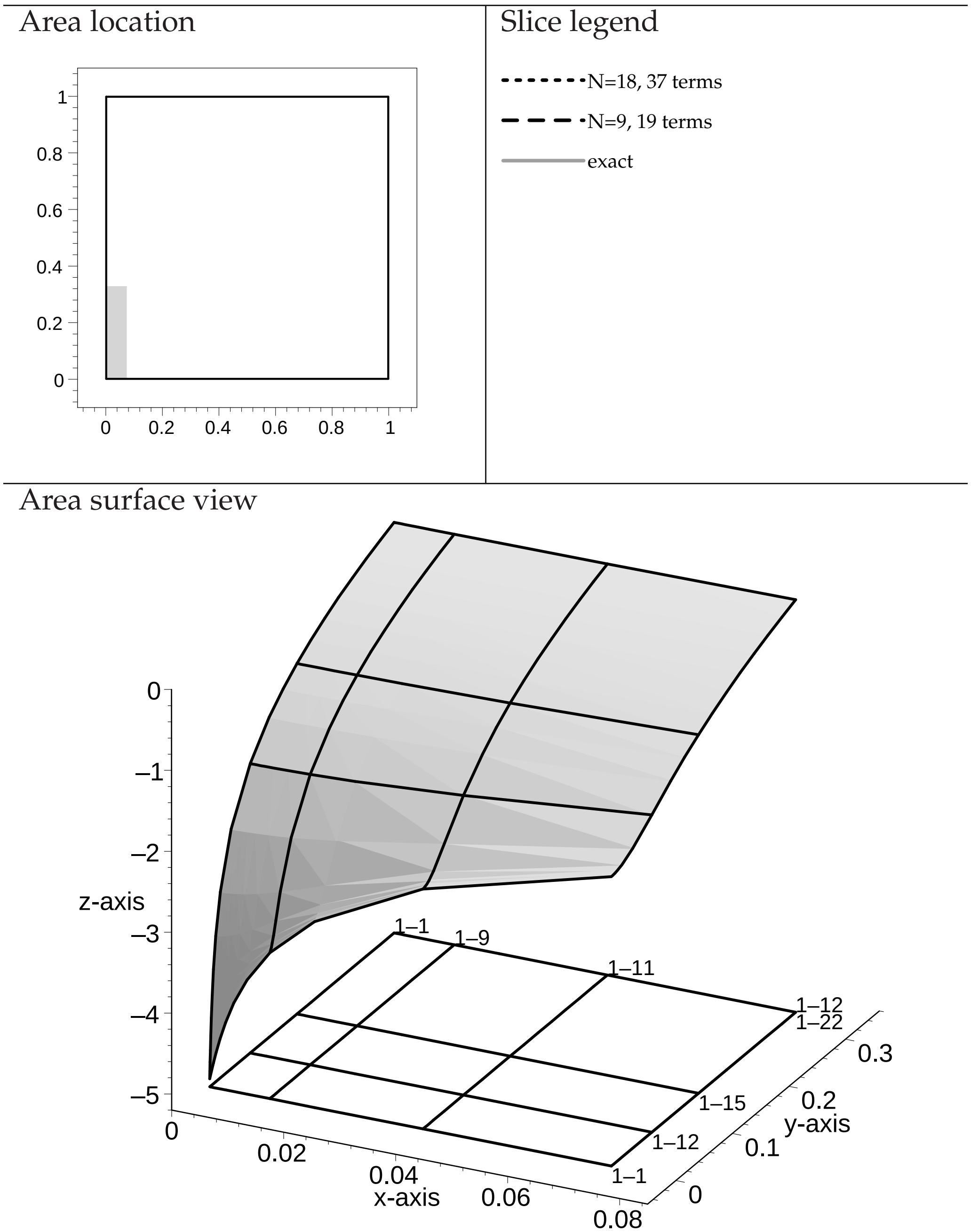


Figure 2.9. Graph displaying u_x as surface, the locations of slices, and the legends of the detailed pointwise error graphs shown in Figure 2.8.

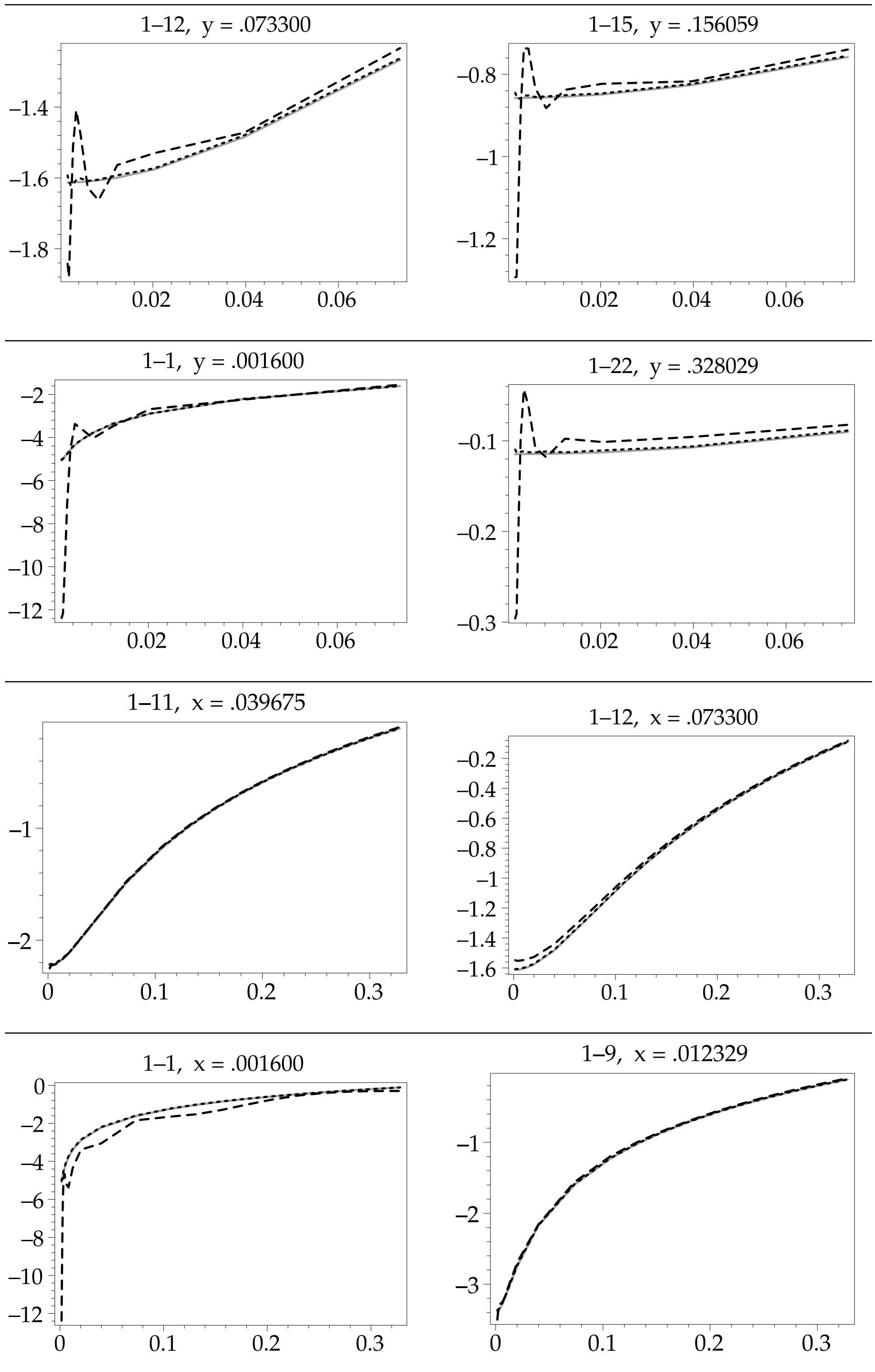


Figure 2.10. Graphs of slices detailing the pointwise error in the approximation of u_x. The locations of these slices are shown, by index, in Figure 2.9.

$m = 7$ and $m = 19$. This behavior is due to the sinc basis functions, and only occurs for derivative approximations; function approximations are uniformly accurate.

Results for u_y are similar and thus omitted.

CHAPTER 3

FORMULATION OF PROBLEMS

One basic idea in fracture mechanics is that the presence of microscopic cracks in materials weakens the material so that material failure occurs at much lower tensile loads than would be expected from other measured material properties. The theory put forth by (Griffith 1924) suggests that the bounding surfaces of solids have surface tension analogous to liquids, and when the strain energy of the material near the crack tip exceeds the potential energy of the surface tension, the crack expands.

The simplest two-dimensional model problems for this type of failure are rectangular plates containing a single crack parallel to one axis. In a proper physical problem, the material plate would be loaded at either the top or bottom boundary and held at the other. In linear fracture mechanics, such problems are usually split into two parts. The particular solution is the solution to the problem of a plate without any crack, loaded and held at the boundaries. The complementary solution is the solution of a plate containing a crack under hydrostatic pressure from the inside, with no loads on the outer boundaries. Once the complimentary solution is obtained, many problems can be solved through the use of different particular solutions.

Two prototypical fracture mechanics problems are re-visited here, a crack in a single-material plate, and a crack between two bonded plates of dissimilar materials. Both are formulated as complimentary problems to allow better comparison to known exact solutions.

3.1 Single-material crack

The geometry of the first model problem is shown in Figure 3.1: a finite, rectangular single-materical plate containing a crack along the horizontal axis. Because of the symmetry of this problem, only the two grey rectangles, labeled 1 and 2, need to be considered numerically. Corresponding to this geometry we have the boundary conditions shown in Figure 3.2, using Einstein summation notation throughout to simplify notation. On the rectangles, the Navier equations have to be satisfied; these are not shown explicitly in the picture, but are derived later.

Referring to Figure 3.2, the boundaries with physically meaningful conditions are the top and right boundaries, as well as the bottom boundary of rectangle 1. On the top and right boundaries, the displacement (v^1, v^2) is zero, while on the bottom boundary of rectangle 1 there is a uniform upward pressure of magnitude σ. This is expressed in standard tensor form: a surface normal vector $\overline{\mathbf{n}}$ is given by $\overline{n}^i, i = 1 \ldots 3$, and the traction vector $\mathbf{T}$ is expressed in terms of the unit normal vector $\mathbf{n}$ and the stress tensor τ.

The remaining boundary conditions have nonphysical meanings. The left boundary of rectangle 1 has a symmetry condition: no horizontal displacement ($v^1 = 0$) and vertical displacement symmetric about the vertical axis ($\frac{\partial v^2}{\partial \theta^1} = 0$). The bottom boundary of rectangle 2 has another symmetry condition: no vertical displacement ($v^2 = 0$) and horizontal displacement symmetric about the horizontal axis ($\frac{\partial v^1}{\partial \theta^2} = 0$). The shared center boundary has a mathematical continuity requirement and the equations in Figure 3.2 have to be interpreted as sums. Thus, these conditions are to be read as

$$ {}^1v^1 - {}^2v^1 = 0 \tag{3.1} $$

$$ {}^1v^2 - {}^2v^2 = 0 \tag{3.2} $$

$$ \frac{\partial^1 v^1}{\partial \theta^1} - \frac{\partial^2 v^1}{\partial \theta^1} = 0 \tag{3.3} $$

$$ \frac{\partial^1 v^2}{\partial \theta^1} - \frac{\partial^2 v^2}{\partial \theta^1} = 0 \tag{3.4} $$

$$ \tag{3.5} $$

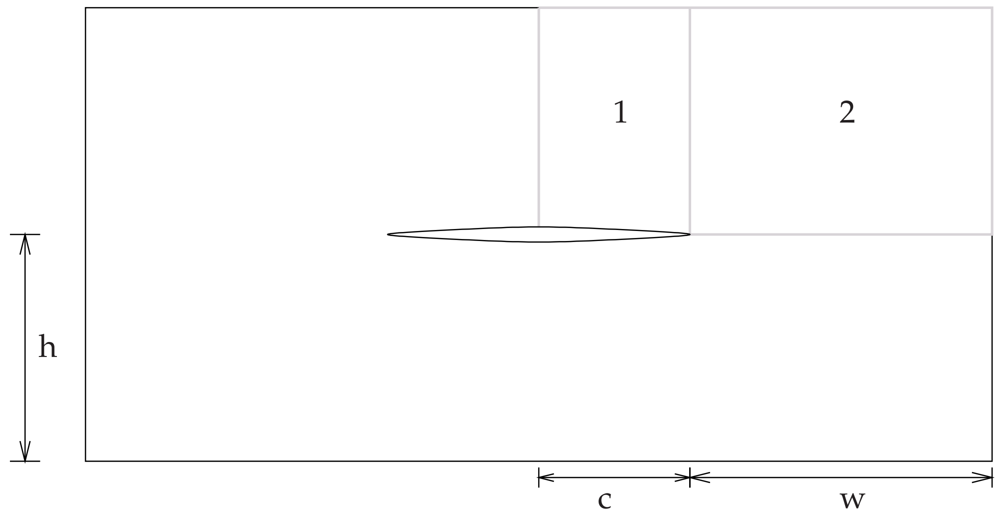


Figure 3.1. Geometry of the single-material crack problem. By symmetry, only the rectangles outlined in grey are needed for numerics; the two computational domains are needed to handle the change in boundary condition from crack surface to the top/bottom material interface. See Figure 3.2 for detailed boundary conditions.

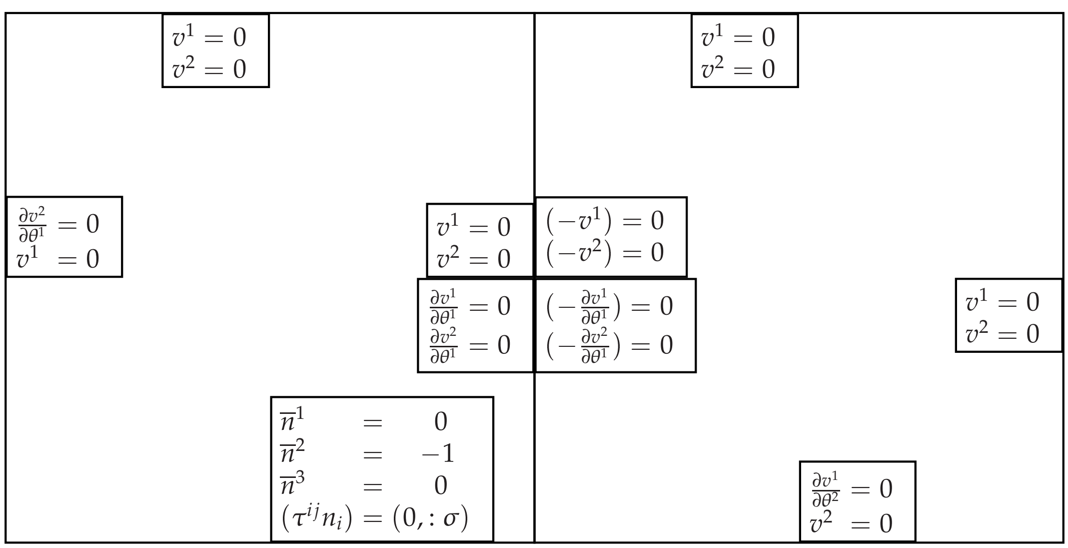


Figure 3.2. The boundary conditions of the top-right part of the single-material crack problem. By horizontal and vertical symmetry, only these two rectangles are needed. These boundary conditions describe a plate fixed along its outside boundary, with uniform pressure applied to the crack surface. See Figure 3.1 for the full geometry. This picture is automatically generated along with input to the numerical solver. Variables of the form $:v$ are global.

where leading superscripts indicate the rectangle.

3.2 Bimaterial crack

The geometry of the second model problem is shown in Figure 3.3: two bonded, finite, rectangular plates with a crack running along their interface. Because of the symmetry of this problem, only the four numbered rectangles, labeled 1 through 4, need to be considered numerically. Corresponding to this geometry we have the boundary conditions shown in Figure 3.4; on the rectangles, the Navier equations have to be satisfied, and these are derived in Section 3.3.

The boundaries with physically-meaningful conditions are the top, right, and bottom boundaries, as well as the bottom boundary of rectangle 1 and the top boundary of rectangle 4. On the top, right, and bottom boundaries, the displacement (v^1, v^2) is zero, while on the bottom boundary of rectangle 1 there is a uniform normal (upward) pressure of magnitude σ, and on the top boundary of rectangle 4 there is a uniform normal (downward) pressure of magnitude σ.

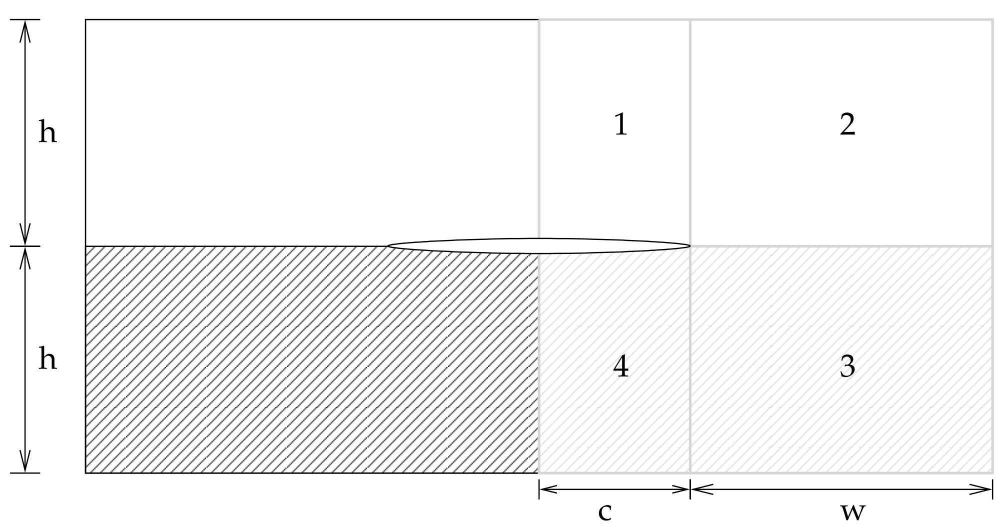


Figure 3.3. Geometry of the bimaterial problem. By symmetry, only the labeled domains are needed for numerics; the 4 computational domains are needed to handle the change in boundary conditions and material parameters. See Figure 3.4 for the boundary conditions.

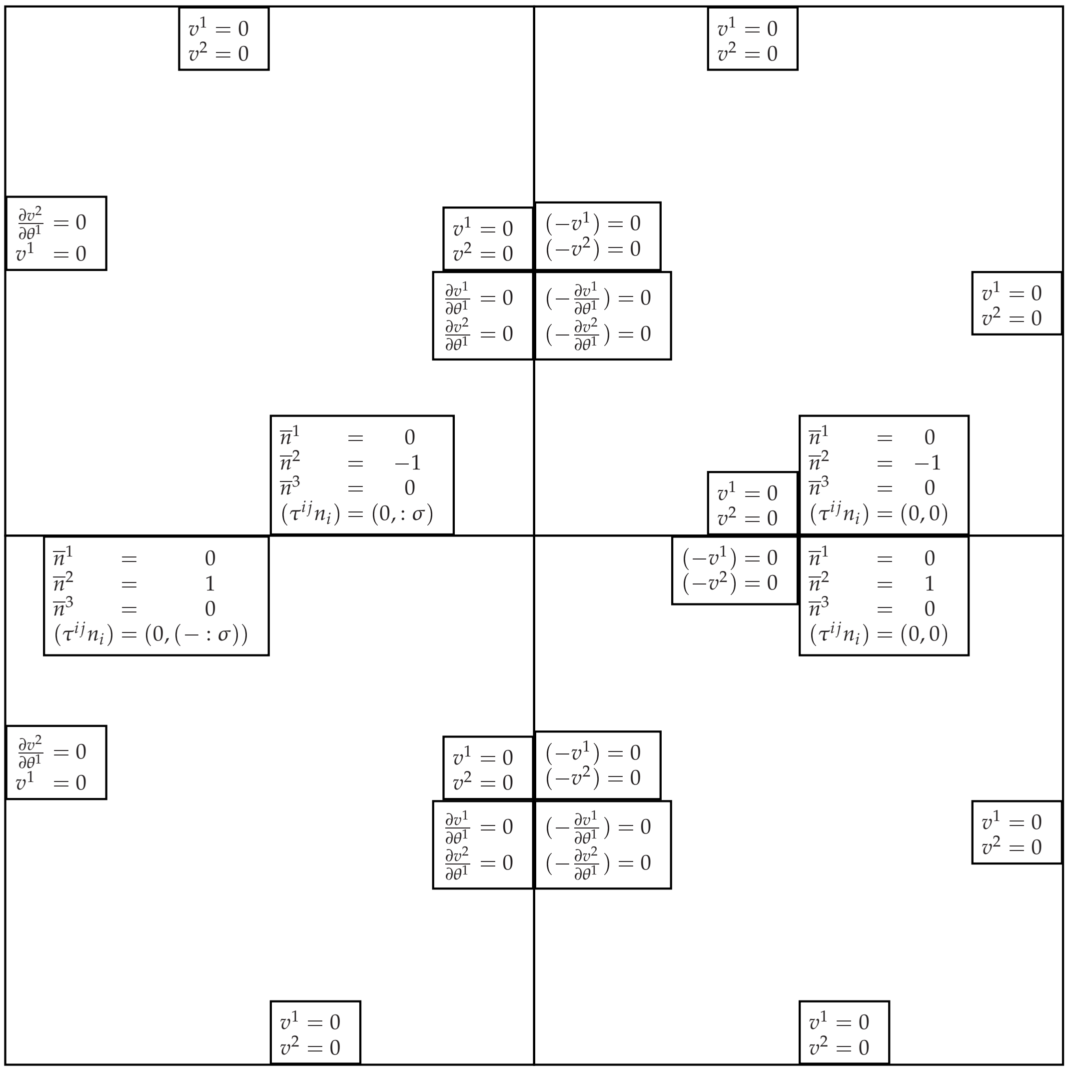


Figure 3.4. Boundary conditions for the bimaterial crack problem. By horizontal symmetry, only these four rectangles are needed. These boundary conditions describe a plate fixed along its outside boundary, with uniform pressure applied to the crack surface. See Figure 3.3 for the full geometry. This picture is automatically generated from input to the numerical solver. Variables of the form $:v$ are global.

Again, these pressures are expressed in standard tensor form: a surface normal vector $\overline{\mathbf{n}}$ is given by $\overline{n}^i, i = 1 \ldots 3$, and the traction vector $\mathbf{T}$ is expressed in terms of the unit normal vector $\mathbf{n}$ and the stress tensor τ.

The remaining boundary conditions have other meanings, analogous to the single-crack problem. The left boundary (rectangles 1 and 4) again has a symmetry condition, and the vertical boundaries in the center again have a mathematical continuity requirement where the equations in Figure 3.4 have to be interpreted as sums. The bottom boundary of rectangle 2 and the top boundary of rectangle 4 fulfill the mechanical continuity requirements. Continuity of the displacement field is ensured through the condition ${}^1v^i - {}^2v^i = 0, i = 1 \ldots 2$, while continuity of the stress field is ensured through the condition ${}^1T^i - {}^2T^i = 0, i = 1 \ldots 2$ on the traction vector $\mathbf{T}$.

3.3 General elasticity equations

The isotropic elasticity equations for a general coordinate system are shown next. Full derivations of these equations, as well as those for more general formulations, are found in (Green and Zerna 1992).

To provide a fully-specified problem for the numerical solver, complete expressions for stresses and the Navier equations are needed in terms of the displacements. These expressions are given in many books for a rectangular Cartesian coordinate system (RCC), and this is in fact the system used here for the two crack problems.

For at least three reasons, however, the general coordinate equations are used here. First, problems may be defined on nonrectangular domains. The SINC-ELLPDE numerical method works on rectangles. Therefore, problems defined on domains bounded by analytical curves must be mapped to a rectangle, and this in turn requires fully general equations. Second, alternative formulations of crack problems may yield better numerical results. The usual modern formulations of crack problems use simple coordinate systems, either to keep the equations simple when obtaining closed-form solutions, or to satisfy restrictions of the numerical method. This is a mistake for the sinc numerical methods which do not distinguish between simple and complex input equations and work on any region mapped to a rectangle, but which are sensitive to the reentrant corners created through the use of rectangles to describe a crack. Third, it is important to emphasize that any valid first-order system can be solved. We therefore provide the equations in a form easily processed by machine, but not usually encountered in the literature.

The original problem solved by (Inglis 1913) involves two concentric ellipses, and the crack as used by (Griffith 1924) is the limiting case using an inner ellipse of zero eccentricity. The SINC-ELLPDE handles boundary layers very well; an elliptical hole formulation with an extremely flat ellipse should yield excellent numerical results. A (short) future project will be to verify this hypothesis.

All input equations for the numerical core are derived from their isotropic tensor forms; basic equations required for expansion are shown in the following, extracted and simplified from the symbolic manipulation program which uses them. This full programmatic implementation is shown in Appendix A.

In general coordinate systems, a distinction is made between two ways of writing one quantity. The covariant base vectors are tangent to the coordinate curves, while the contravariant base vectors are orthogonal to the covariant vectors. Every tensor can be written using covariant components, indicated by the use of subscripts, or using contravariant components, indicated by the use of superscripts. To simplify notation, Einstein summation notation is used throughout. Thus, indices repeated in terms of a product are implicitly summed over unless otherwise indicated.

3.3.1 Coordinate system transformation

In the following, the x^i coordinate system is assumed to be rectangular Cartesian, so $x^i = x_i$. The elasticity equations are written for the θ^i system, which can be an arbitrary coordinate system for which the transformation to the x^i system is given by

$$x^i = x^i(\theta^1, \theta^2, \theta^3)$$

3.3.2 Metric tensors

Once the coordinate transformation is defined, certain quantities are needed repeatedly in transformation. The first of these is the metric tensor, given by

$$g_{ij} = \frac{\partial x^m}{\partial \theta^i}\frac{\partial x^m}{\partial \theta^j}.$$

3.3.3 Christoffel symbols

The simple partial derivatives in rectangular Cartesian coordinates become covariant derivatives in general systems. In the computation of covariant derivatives, Christoffel symbols are needed. They are given by

$$\begin{bmatrix} i, j\,k \end{bmatrix} = \frac{1}{2}\left(\frac{\partial g_{ij}}{\partial \theta^k} + \frac{\partial g_{ik}}{\partial \theta^j} - \frac{\partial g_{jk}}{\partial \theta^i}\right)$$

and

$$\begin{Bmatrix} i \\ j \;\; k \end{Bmatrix} = g^{il}\begin{bmatrix} l, j\,k \end{bmatrix}$$

3.3.4 Covariant derivatives

For the Navier equations, only the derivatives for the v^i and $T^i{}_k$ components are needed. These are given by

$$v^r{}_i = \frac{\partial v^r}{\partial \theta^i} + \begin{Bmatrix} r \\ s \;\; i \end{Bmatrix} v^s$$

and

$$v^i{}_{kl} = \frac{\partial v^i{}_k}{\partial \theta^l} + \begin{Bmatrix} i \\ m \;\; l \end{Bmatrix} v^m{}_k - \begin{Bmatrix} m \\ k \;\; l \end{Bmatrix} v^i{}_m$$

3.3.5 Navier equations

Using the previously defined quantities, the Navier equations without body-force terms can be written as

$$(1-2\nu)g^{sj}v^i{}_{sj} + g^{is}v^j{}_{js} = 0 \qquad \text{for } i = 1\ldots 3 \tag{3.6}$$

3.3.6 Displacement boundary conditions

The specification of displacement boundary conditions is slightly more involved in general coordinates than in a rectangular Cartesian system. The physical displacement components are usually the known ones ($\overline{D}^i$), and these must first be converted to their tensor equivalents, D^i, via the formula

$$D^i = \frac{\overline{D}^i}{\sqrt{g_{ii}}}$$

Once the displacement tensor components D^i are known, they can be directly used in boundary conditions via the equation

$$D^i = v^i$$

where $\mathbf{v}$ is the unknown displacement, $\mathbf{D}$ the specified. After a displacement solution in tensor form is obtained (D^i), the physical displacement components are typically needed again for viewing. The (trivial) inverse is therefore also useful:

$$\overline{D}^i = \sqrt{g_{ii}} D^i$$

3.3.7 Traction boundary conditions

The specification of traction boundary conditions is slightly more involved in general coordinates than in a rectangular Cartesian system. First, the unit normal vector to the boundary is needed. It should be straightforward to find a normal vector, and using the equation

$$n^i = \frac{\overline{n}^i}{\sqrt{g_{kj}\overline{n}^k\overline{n}^j}}$$

the unit vector's components are calculated from the $\overline{\mathbf{n}}$ vector's components. Secondly, the physical (applied) traction will typically be known, and this is converted to the proper traction vector via

$$T^i = \frac{\overline{T}^i}{\sqrt{g_{ii}}}$$

Using these quantities, traction boundary conditions can now be specified via

$$T^j = \tau^{ij} n_i$$

where T^i are the proper tensor components of the specified boundary conditions, **n** is the normal vector to the boundary, here expressed via covariant components given by

$$n_i = g_{ij} n^j$$

and the connection to the final unknowns v^i is given by the stress-displacement conditions

$$\tau^{ij} = \mu\left(g^{js} v^i{}_s + g^{is} v^j{}_s + \frac{2\nu}{(1-2\nu)} g^{ij} v^s{}_s\right)$$

3.4 Expanded elasticity equations

For the present problems, only a rectangular Cartesian system is needed; the mapping is the simplest possible,

$$x^i = \theta^i$$

The final expanded forms of the equations shown in Section 3.3 follow. These are extracted and simplified versions of the full program's output shown in Appendix B.

3.4.1 Metric tensors

$$g_{ij} = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3.7}$$

$$g^{ij} = \delta_{ij} \tag{3.8}$$

3.4.2 Christoffel symbols

$$\begin{bmatrix} i, j\ k \end{bmatrix} = 0 \tag{3.9}$$

$$\begin{Bmatrix} i \\ j\ \ k \end{Bmatrix} = 0 \tag{3.10}$$

3.4.3 The full Navier equations

$$(2-2\nu)\frac{\partial^2 \mathrm{u}^1}{(\partial\theta^1)^2}+(1-2\nu)\frac{\partial^2 \mathrm{u}^1}{(\partial\theta^2)^2}+(1-2\nu)\frac{\partial^2 \mathrm{u}^1}{(\partial\theta^3)^2}+\frac{\partial^2 \mathrm{u}^3}{\partial\theta^1\partial\theta^3}+\frac{\partial^2 \mathrm{u}^2}{\partial\theta^1\partial\theta^2}=0$$

$$(1-2\nu)\frac{\partial^2 \mathrm{u}^2}{(\partial\theta^1)^2}+(2-2\nu)\frac{\partial^2 \mathrm{u}^2}{(\partial\theta^2)^2}+(1-2\nu)\frac{\partial^2 \mathrm{u}^2}{(\partial\theta^3)^2}+\frac{\partial^2 \mathrm{u}^3}{\partial\theta^2\partial\theta^3}+\frac{\partial^2 \mathrm{u}^1}{\partial\theta^1\partial\theta^2}=0 \tag{3.11}$$

$$(1-2\nu)\frac{\partial^2 \mathrm{u}^3}{(\partial\theta^1)^2}+(1-2\nu)\frac{\partial^2 \mathrm{u}^3}{(\partial\theta^2)^2}+(2-2\nu)\frac{\partial^2 \mathrm{u}^3}{(\partial\theta^3)^2}+\frac{\partial^2 \mathrm{u}^2}{\partial\theta^2\partial\theta^3}+\frac{\partial^2 \mathrm{u}^1}{\partial\theta^1\partial\theta^3}=0$$

3.4.4 The full stress tensor

The components of τ^{ij} are

$$\begin{aligned}
[1,2] &= \mu\frac{\partial \mathrm{u}^1}{\partial\theta^2}+\mu\frac{\partial \mathrm{u}^2}{\partial\theta^1}\\
[2,2] &= \frac{2\mu(-1+\nu)}{-1+2\nu}\frac{\partial \mathrm{u}^2}{\partial\theta^2}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^1}{\partial\theta^1}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^3}{\partial\theta^3}\\
[1,3] &= \mu\frac{\partial \mathrm{u}^1}{\partial\theta^3}+\mu\frac{\partial \mathrm{u}^3}{\partial\theta^1}\\
[1,1] &= \frac{2\mu(-1+\nu)}{-1+2\nu}\frac{\partial \mathrm{u}^1}{\partial\theta^1}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^2}{\partial\theta^2}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^3}{\partial\theta^3}\\
[3,2] &= \mu\frac{\partial \mathrm{u}^2}{\partial\theta^3}+\mu\frac{\partial \mathrm{u}^3}{\partial\theta^2}\\
[2,3] &= \mu\frac{\partial \mathrm{u}^2}{\partial\theta^3}+\mu\frac{\partial \mathrm{u}^3}{\partial\theta^2}\\
[2,1] &= \mu\frac{\partial \mathrm{u}^1}{\partial\theta^2}+\mu\frac{\partial \mathrm{u}^2}{\partial\theta^1}\\
[3,3] &= \frac{2\mu(-1+\nu)}{-1+2\nu}\frac{\partial \mathrm{u}^3}{\partial\theta^3}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^1}{\partial\theta^1}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathrm{u}^2}{\partial\theta^2}\\
[3,1] &= \mu\frac{\partial \mathrm{u}^1}{\partial\theta^3}+\mu\frac{\partial \mathrm{u}^3}{\partial\theta^1}
\end{aligned} \tag{3.12}$$

3.4.5 Two-dimensional Navier equations

The full three-dimensional equations were used in the preceding sections for simplicity and generality, but the two-dimensional equations are needed for the numerical algorithm. For planar elasticity problems, the two standard choices are plane-stress and plane-strain; here, plane-strain assumptions are used for simplicity. Using these assumptions and removing all z direction components (θ^3), the following are the interior equations in all domains.

$$\begin{aligned}(-2\nu+2)\frac{\partial^2 \mathsf{u}^1}{(\partial\theta^1)^2}+(1-2\nu)\frac{\partial^2 \mathsf{u}^1}{(\partial\theta^2)^2}+\frac{\partial^2 \mathsf{u}^2}{\partial\theta^1\partial\theta^2} &= 0\\ (1-2\nu)\frac{\partial^2 \mathsf{u}^2}{(\partial\theta^1)^2}+(-2\nu+2)\frac{\partial^2 \mathsf{u}^2}{(\partial\theta^2)^2}+\frac{\partial^2 \mathsf{u}^1}{\partial\theta^1\partial\theta^2} &= 0\end{aligned} \tag{3.13}$$

3.4.6 Two-dimensional stress tensor

Again using plane-strain assumptions, the components of the two-dimensional τ^{ij} are given by

$$\begin{aligned}[1,2] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^2}+\mu\frac{\partial \mathsf{u}^2}{\partial\theta^1}\\ [2,2] &= \frac{-2\mu(1-\nu)}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial\theta^1}\\ [1,1] &= \frac{2\mu(-1+\nu)}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial\theta^1}-\frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2}\\ [2,1] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^2}+\mu\frac{\partial \mathsf{u}^2}{\partial\theta^1}\end{aligned} \tag{3.14}$$

3.5 Mathematical view

From a mathematical point of view, both crack problems are linear second-order elliptic systems in two unknowns defined on a union of rectangles, with analytical (constant, in fact) coefficients on each rectangle. As such, both can be easily converted to first-order systems and handled by the SINC-ELLPDE algorithm. Specifically, the traction and displacement boundary conditions are already in proper form, while the Navier equations are rewritten as[1]

[1] For historical reasons, the second-order displacement formulation of the Navier equations is used here. This is converted to a first-order system through the introduction of new unknowns.

$$
\begin{aligned}
(-2\nu+2)\,\frac{\partial \mathsf{u}^{1,1}}{\partial \theta^1} + (1-2\nu)\,\frac{\partial \mathsf{u}^{1,2}}{\partial \theta^2} + \frac{\partial \mathsf{u}^{2,1}}{\partial \theta^2} &= 0, \\
(1-2\nu)\,\frac{\partial \mathsf{u}^{2,1}}{\partial \theta^1} + (-2\nu+2)\,\frac{\partial \mathsf{u}^{2,2}}{\partial \theta^2} + \frac{\partial \mathsf{u}^{1,1}}{\partial \theta^2} &= 0 \\
\frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}^{1,1} &= 0 \\
\frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}^{1,2} &= 0 \\
\frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}^{2,1} &= 0 \\
\frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}^{2,2} &= 0
\end{aligned}
\tag{3.15}
$$

It is important to note that anisotropic materials and nonrectangular coordinate systems can be used by the SINC-ELLPDE algorithm without problems, but the input equations would be substantially more complex. Problems of this type, as solved in (Clements 1971; Li and Nemat-Nasser 1990) may be addressed in the future.

CHAPTER 4

METHOD OF SOLUTION

The class of problems considered here are two-dimensional linear elliptic first-order systems of partial differential equations and their boundary conditions, with or without corner and/or edge singularities, defined on a finite connected union of rectangles.

On each rectangle, the unknowns' coefficients and the solution are assumed to be smooth; using multiple rectangles, piecewise smooth systems can be solved.

The method of solution is based on the collocation of a sinc-series representation of the first-order systems, and is henceforth referred to as the SINC-ELLPDE method. This method is based on one-dimensional results proven in (Stenger 1993a; Morlet et al. 1997), extended for the present class of problems; this extension involves significant modifications and simplifications of the notation to allow concise expression of the method.

The approaches in (Morlet et al. 1997) and in Section 7.2.6 of (Stenger 1993a) for handling nonhomogeneous derivative boundary conditions involve manual selection of appropriate basis functions and splines, and lead to a very complicated two-dimensional algorithm, described in Appendix C. An implementation of that algorithm was written and found to be too complex for practical use: it required manually selecting the appropriate basis functions for two directions of every unknown in every rectangle. The algorithm also had two defects: the solutions of the equations had to have bounded derivatives, which excludes crack problems, and the algorithm produced a large, dense matrix.

Here, a simplified approach is used. Mathematical background information is provided in three sections. Section 4.1 presents definitions relevant to the current algorithm; needed theorems are in Section 4.2. These mostly follow the concise summary of basic one-dimensional sinc properties found in (Stenger 1993c). The practical use of these results for sinc collocation is derived for a single elliptic operator and a single unknown in one dimension in Section 4.3. The notational extensions needed for multiple unknowns/operators are shown in Section 4.4, followed by the main topic of this chapter, the general SINC-ELLPDE method, in Section 4.5.

The steps of the method are detailed in Sections 4.5.1 through 4.5.5, accompanied by a complete, relatively simple abstract problem. This problem is similar to that of Chapter 2, but is presented to illustrate the method, not its use in practice.

4.1 Basic definitions

Definition 4.1 ($\mathcal{D}_d, \mathcal{D}, \phi, \psi$) *Pick $d > 0$ and define the strip $\mathcal{D}_d$ by*

$$\mathcal{D}_d = \{w \in \mathbb{C} : |\Im w| < d\} \tag{4.1}$$

Given a region $\mathcal{D}$ containing a contour Γ in $\mathbb{C}$ with endpoints a and b on the boundary of $\mathcal{D}$, ϕ is a one-to-one conformal map with the properties $\phi : \Gamma \to R$, $\phi(x) \to -\infty$ as $x \to a$, $\phi(x) \to \infty$ as $x \to b$ and $\phi : \mathcal{D} \to \mathcal{D}_d$

Define $\psi \equiv \phi^{-1}$; then the region $\mathcal{D}$ is the image

$$\mathcal{D} = \psi(\mathcal{D}_d) \tag{4.2}$$

Figure 4.1 shows a typical example for the functions $\phi(x) = \ln(x-a)/(b-x)$ and $\psi(z) = (\exp(z)b + a)/(\exp(z)+1)$ using values $d = \pi/3$, $a = 1$, and $b = 3$. These functions are also the ones used in the remainder of this work. Many other maps are available; see (Stenger 1993a), Section 1.7.

Definition 4.2 (ρ, L_α) *For the region $S_d = \{z \in \mathbb{C} : |\arg z| < d\}$, the map $\rho : \mathcal{D} \to S_d$ is defined as*

$$\rho(z) = e^{\phi(z)} \tag{4.3}$$

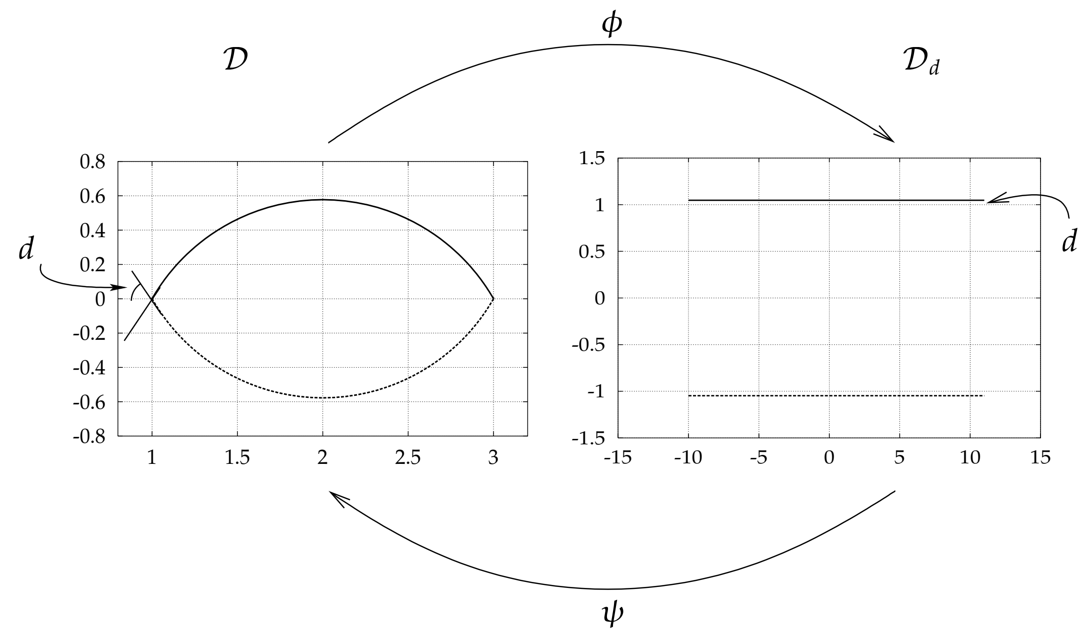


Figure 4.1. A part of the strip-shaped infinite region $\mathcal{D}_d$ and its image $\mathcal{D}$, for $\phi(x) = \ln(x-a)/(b-x)$, $\psi(z) = (\exp(z)b + a)/(\exp(z)+1)$, $d = \pi/3$, $a = 1$, $b = 3$. The points $kh : k \in Z$ are mapped to $[a, b]$.

Notice that for $z \in \mathbb{R}$, $\rho : \Gamma \to [0, \infty)$. Given $\alpha > 0$, $d > 0$ and a region $\mathcal{D}$, denote by L_α the family of all functions $F(z)$ analytic and uniformly bounded in $\mathcal{D}$ so that $\forall z \in \mathcal{D}$,

$$|F(z)| \le \frac{C|\rho(z)|^\alpha}{|1+\rho(z)|^{2\alpha}} \tag{4.4}$$

for some $C > 0$.

On $\mathbb{R}$ using $\phi(z) = z$, this criterion is

$$|F(z)| \le \frac{C|e^z|^\alpha}{|1+e^z|^{2\alpha}} \tag{4.5}$$

so as $z \to \infty$, $|F(z)| \le C_1|e^{-\alpha z}|$ and as $z \to -\infty$, $|F(z)| \le C_2|e^{\alpha z}|$ and L_α is the class of exponentially decaying functions.

On $[a, b]$ using $\phi(z) = \ln(z-a)/(b-z)$,

$$|F(z)| \le C\left|\frac{z-a}{b-z}\right|^\alpha \left|\frac{b-z}{b-a}\right|^{2\alpha} \tag{4.6}$$

so as $z \to a^+$, $|F(z)| \le C_1|z-a|^\alpha$ and as $z \to b^-$, $|F(z)| \le C_2|b-z|^\alpha$, so algebraic decay near the endpoints is required of L_α functions in this case.

Definition 4.3 (M_α) *Let $\alpha \in (0, 1]$ and $d \in (0, \pi)$. Define f as*

$$f = g - \left[\frac{(b-z)g(a) + (z-a)g(b)}{b-a}\right] \tag{4.7}$$

Then $M_\alpha(\mathcal{D})$ denotes the family of functions g analytic and uniformly bounded in $\mathcal{D}$ such that $f \in L_\alpha(\mathcal{D})$

Functions in the $M_\alpha(\mathcal{D})$ class may have nonzero values at the endpoints, and this class is used in the remainder of this work.

Next are the basic elements used for approximation on $\mathbb{R}$.

Definition 4.4 (Approximation on $\mathbb{R}$) *The sinc function is defined by*

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \tag{4.8}$$

For a given $N > 0$, the j^{th} sinc point is given by

$$z_j = \phi^{-1}(jh) = \psi(jh) \tag{4.9}$$

where the sinc spacing parameter is

$$h = \sqrt{\frac{\pi d}{\alpha N}} \tag{4.10}$$

Given $N > 0$ *the* k^{th} *sinc series term on* $\mathbb{R}$ *is defined by*

$$\eta_k(x) = \begin{cases} t_L(x) - \sum_{j=-N+1}^{N} t_L(jh)S(j,x) & k = -N \\ S(k,x) & k \in -N+1..N-1 \\ t_R(x) - \sum_{j=-N}^{N-1} t_R(jh)S(j,x) & k = N \end{cases} \tag{4.11}$$

with

$$t_L(x) = \frac{1}{1+e^x} \tag{4.12}$$

$$S(j,x) = \operatorname{sinc}\left(\frac{x-jh}{h}\right) \tag{4.13}$$

$$t_R(x) = \frac{e^x}{1+e^x} \tag{4.14}$$

These definitions are used on a contour Γ via the appropriate conformal map ϕ and the new functions $\omega_k(z)$, defined as follows.

Definition 4.5 (Approximation on Γ) *The* k^{th} *sinc series term on* Γ *is defined by*

$$\omega_k(z) = \eta_k(\phi(z)).$$

4.2 Known one-dimensional properties of sinc series

As mentioned in the introduction, sinc series can be used for numerical approximation of many calculus operations. Here, those theorems relevant for collocation of partial differential equations are repeated. Full proofs can be found in (Stenger 1993a). In the following, the norm $\|\cdot\|$ is the maximum norm on Γ, i.e., for $f(z) \in \Gamma$,

$$\|f(z)\| = \max_{z\in\Gamma}|f(z)| \tag{4.15}$$

and the $(2N+1)$-term approximation error is given by

$$\epsilon_N = \sqrt{N}e^{-\sqrt{\pi d\alpha N}} \tag{4.16}$$

Theorem 4.6 (Interpolation) *If $f \in M_\alpha(\mathcal{D})$, then there exists a constant $C > 0$, independent of N, s.t.*

$$\|f - \sum_{j=-N}^{N} f(z_j)\omega_j\| \le C\epsilon_N \tag{4.17}$$

Theorem 4.7 (Differentiation) *Let $f \in M_\alpha(\mathcal{D}')$, let μ be any nonnegative integer and if $\mu > 1$, let $(1/\phi')'$ be uniformly bounded in $\mathcal{D}'$. Then there exists a constant C_μ independent of N such that*

$$\left\| \left(\frac{h}{\phi'}\right)^{(k)} \left[f - \sum_{j=-N}^{N} f(z_j)\omega_j \right]^{(k)} \right\| \le C_\mu \epsilon_N \tag{4.18}$$

for k = 0,1, . . . , μ.

Theorem 4.8 (Collocation) *Let $f \in M_\alpha(\mathcal{D})$. Let $\mathbf{c} = (c_{-N}, \ldots, c_N)^T$ be a complex vector such that*

$$\left(\sum_{j=-N}^{N} |f(z_j) - c_j|^2 \right)^{1/2} < \delta \tag{4.19}$$

where $\delta > 0$. Then

$$\|f - \sum_{j=-N}^{N} c_j\omega_j\| < C\epsilon_N + \delta, \tag{4.20}$$

with C as in Equation 4.17.

4.3 Per-unknown errors in collocation

This section combines the previously shown results and presents a short overview of the general steps used in sinc-collocation.

Given an invertible linear elliptic differential operator L and the linear system

$$Lu = f, \tag{4.21}$$

define the vector operator $[u]$ as $[u] = (u(z_{-N}), \ldots, u(z_N))^T$ and the matrix $[L]$ as

$$[L]_{ij} = (L\omega_j)(z_i) \tag{4.22}$$

Any stable solution method then obtains a vector $\mathbf{c}$ satisfying

$$[L]\mathbf{c} = [f] + [\epsilon_u] \tag{4.23}$$

with ϵ_u proportional to the unit-roundoff error. Define $\mathbf{u} = [u]$. From Theorem 4.7, obtain

$$[Lu] = [L]\mathbf{u} + [\epsilon_N] \tag{4.24}$$

Now,

$$[L](\mathbf{u} - \mathbf{c}) = [L]\mathbf{u} - ([f] + [\epsilon_u]) \tag{4.25}$$

and therefore

$$\|\mathbf{u} - \mathbf{c}\| \leq \|[L]^{-1}\| \left\{ \Big\|[L]\mathbf{u} - [Lu]\Big\| + \Big\|[Lu] - [f]\Big\| + \Big\|[\epsilon_u]\Big\| \right\} \tag{4.26}$$

$$= \|[L]^{-1}\| \left\{ \Big\|[\epsilon_N]\Big\| + \Big\|[0]\Big\| + \Big\|[\epsilon_u]\Big\| \right\}. \tag{4.27}$$

As long as $[L]$ is invertible, $u \in M_\alpha$, and $\|[L]^{-1}\|$ is sufficiently small, the computed vector $\mathbf{c}$ satisfies Theorem 4.8 and the solution u can be uniformly computed via

$$u \approx \sum_{j=-N}^{N} c_j \omega_j, \tag{4.28}$$

with error bound given by Equation 4.20.

In Lemma 7.2.5, Section 7.2 of (Stenger 1993a), the bound $\|[L]^{-1}\| = O(N^2)$ is derived for a second-order linear boundary-value problem under suitable conditions on the coefficients.

4.4 Components and notation for general problems

By considering more general L, u and f, the derivation of Section 4.3 directly extends to higher dimensions. It also extends to multiple-operator/unknown systems, at the cost of introducing one ϵ_N-size error per unknown.

Given a linear elliptic system with at least one unknown and one domain, e.g., the boundary conditions of Figure 3.2 together with the main equations (3.13), one can obtain the equivalent matrix formulation as follows.

To uniquely label unknowns (and their series terms), domains, equations and regions (and their associated collocation points), introduce the following notation.

Definition 4.9 (Index Notation) *Assuming unknowns and directions are indexed, the notation*

$$i|j,k \tag{4.29}$$

denotes unknown i in domain j and direction k. Usually, i or k will be absent. Similarly, the notation

$$i \wedge R \tag{4.30}$$

denotes equation i in region R

Using this notation, the two-dimensional collocation points, basis functions, operators and right-hand sides are fully described by the following definitions.

Since a given problem consists of main equations and boundary conditions, it is natural to partition its discrete version, specified by the collocation points, into analogous regions. This gives rise to the following definition.

Definition 4.10 (Collocation Points and Regions) *The collocation points are given by $z^{|j}$, the array of points for the j^{th} domain with entries*

$$z^{|j}_{kl} = \psi^{|j,x}(kh^{|j,x}), \psi^{|j,y}(lh^{|j,y}) \tag{4.31}$$

These collocation points are partitioned into regions analogous to the equations and boundary conditions of the problem class. On every rectangular domain, we thus have the regions top (T)*, right* (R)*, left* (L)*, bottom* (B)*, and interior* (I)*.*

To handle multiple domains, we additionally introduce the regions top-, right-, bottom- and left-overlap (T°, R°, B° *and* L°, *respectively). Their locations are identical to the regions* T, R, B *and* L, *respectively, but their meaning and use are different.*

The precise indices of the point regions are shown in Table 4.1, and a graphical illustration in Figure 4.2.

The basis functions are derived as a simple tensor product.

Table 4.1. Points corresponding to regions, by index. There are nine logical point regions of interest for two-dimensional sinc collocation, corresponding to five geometric regions of a rectangle: the top, right, bottom and left boundaries, and the interior. For a pictorial view, see Figure 4.2.

Region	Points $z_{kl}^{\|j}$	
$\mathsf{T}, \mathsf{T}^{\circ}$	$k = -N^{\|j,x} + 1, \ldots, N^{\|j,x} - 1$	$l = N^{\|j,y}$
$\mathsf{R}, \mathsf{R}^{\circ}$	$k = N^{\|j,x}$	$l = -N^{\|j,y} + 1, \ldots, N^{\|j,y} - 1$
$\mathsf{B}, \mathsf{B}^{\circ}$	$k = -N^{\|j,x} + 1, \ldots, N^{\|j,x} - 1$	$l = -N^{\|j,y}$
$\mathsf{L}, \mathsf{L}^{\circ}$	$k = -N^{\|j,x}$	$l = -N^{\|j,y} + 1, \ldots, N^{\|j,y} - 1$
I	All remaining points	

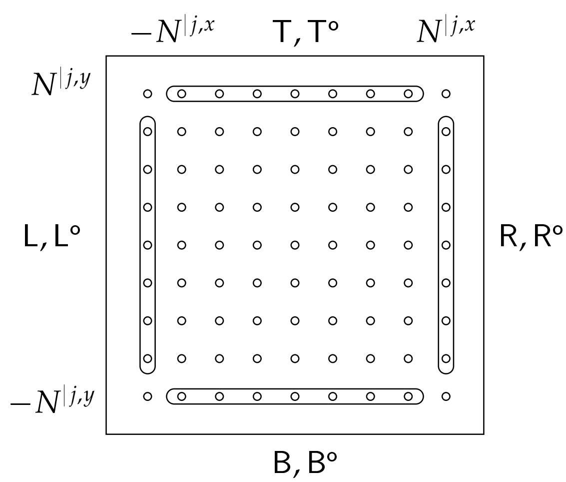


Figure 4.2. A picture illustrating the correspondence of points and logical collocation regions of any rectangle j. The enclosed points belong to each of the indicated regions; all other points, including the four corner points, belong to the interior region I. For the precise range of indices, see Table 4.1.

Definition 4.11 (Basis Functions) *The i^{th} unknown in the j^{th} domain is given by*

$$u^{i|j}(x,y) \;=\; \sum_{k=-N^{|j,x}}^{N^{|j,x}} \sum_{l=-N^{|j,y}}^{N^{|j,y}} c_{kl}^{i|j}\,\omega_k^{|j,x}(x)\,\omega_l^{|j,y}(y) \tag{4.32}$$

where $\omega_k^{|j,x}$ is the k^{th} ω in the x direction in domain j.

The given equations and boundary conditions are viewed as a collection of operators acting on the unknowns, with given right-hand sides.

Definition 4.12 (Operators) *The operator for the i^{th} unknown in the j^{th} equation for point-region R of rectangle k is written as*

$$L_{j\wedge R}^{i|k} \tag{4.33}$$

Definition 4.13 (Right-hand functions) *The right-hand function for the j^{th} equation of region R in domain k is written as*

$$f_{j\wedge \mathsf{R}}^{|k} \tag{4.34}$$

4.5 Method for general problems

Using these definitions, all equations and boundary conditions of a given problem are combined to form a single large linear system which is then discretized; the resulting matrix is solved in one step. The solution of the linear system is obtained via a standard linear system solver, and the individual unknowns' coefficients extracted. Approximations to the individual unknowns (and their derivatives) can then be computed. This can be shown simply as a diagram:

$$\begin{tikzcd}
\text{PDE/BC} \arrow[dr, "\textit{block conversion}"'] & & & & u_i \\
& Lu = f \arrow[dr, "\textit{discretization}"'] & & {[u]} \arrow[ur, "\textit{reconstruction}"] & \\
& & {[L][u] = [f]} \arrow[ur, "\textit{solution}"] & &
\end{tikzcd} \tag{4.35}$$

The transformation algorithms are the topic of Chapter 6 and are not mentioned further here. In the remainder, the original PDE/BC will be referred to as the PDE system, the equation $Lu = f$ as the block system, the equation $[L][u] = [f]$ as the discrete block system, $[u]$ as the discrete approximation and the u_i as the smooth approximations. The focus of this chapter is on these stages of the SINC-ELLPDE method:

PDE system The complete, original set of equations to be solved is referred to as the PDE system. This includes the main equations and all boundary conditions, written in a very precise format. By choosing appropriate notation for individual operators and unknowns, this format enforces proper structure and allows straightforward identification of the blocks constituting the block system. This form shows all differential operators and unknowns, but has no block structure.

block system The entire PDE system is rewritten in the continuous linear operator form $Lu = f$. In this form, only matrix blocks remain; all differential operators are contained in the blocks, and all sums are implied by the blocks' positions in the operator L.

discrete block system Using Theorem 4.7, the block system can be discretized to yield the discrete block system $[L][u] = [f]$. This incurs an error of $O(\epsilon_N)$ for every block.

discrete approximation Solution of the discrete block system yields the vector $[u]$; from Section 4.3, this vector provides good approximate values of u at the sinc points. The error introduced here is determined by the accuracy of the linear system solution, which is determined by the interaction between the individual blocks of $[L]$. No rigorous derivation of the errors introduced in this step exists yet; for single-unknown, two-domain problems coupled in the same manner as the present problems, an $O(n^4\epsilon_N)$ error is derived in (Morlet, Lybeck, and Bowers 1994); this is overly pessimistic, and in practice an $O(n^{1/2}\epsilon_N)$ error is observed. Generally, accuracy of the linear system solution is limited by the conditioning of the matrix $[L]$, and this conditioning is easily checked by experiment.[1]

smooth approximation Via Theorem 4.8, the discrete approximation provides the smooth approximation $\overline{u}$ at any point inside the rectangle, introducing another $O(\epsilon_N)$ error per unknown.

The precise errors accumulated in the solution of a problem are specific to that problem, but the per-unknown interpolation- and collocation errors are always of the same order, as is the error of the solution of the linear system. Because of this, and given the special structure of the systems dealt with, a simple example illustrating the combined errors is sufficient to illustrate how error bounds can be derived for any other system. The following sections use a generic Poisson's equation to illustrate the steps of the method.

4.5.1 PDE system

Borrowing the concept of BNF grammars[2] from computer science allows a very precise specification of allowable equation formats. This format provides a common syntactic framework for *all* problems solvable by the SINC-ELLPDE algorithm. It also clearly identifies all parts of a system of equations, and leads directly to the operator-unknown format used in the following discussions.

It should be noted that this format restriction is purely syntactic; it is quite possible to specify a hyperbolic system using it, which will typically result in a singular matrix $[L]$; attempts to solve an incompletely-specified problem will usually fail in the same way.

---

[1] The reciprocal condition number was found to be in the range $10^{-8} - 10^{-12}$ for the problems considered here; this is well above the unit-roundoff error for IEEE double precision, $\approx 2.22e - 16$.

[2] The full concepts and theory of parsing are discussed at length in (Aho, Sethi, and Ullman 1986), while the practical use of lexing and parsing tools is described in (Levine, Mason, and Brown 1992). Complete examples are also found in (Kernighan and Pike 1984).

The BNF grammar for the problems solvable by the current method (and problems not solvable by it) is shown in Figure 4.3. These rules are derived as follows.

As the systems considered here are linear, every operator has the <expr> syntax shown in Figure 4.3: one algebraic expression multiplying one unknown or derivative thereof. Of course, every equation can have multiple operators acting on one or more unknowns; hence a sum or difference of coefficient-operator-

```
nonterminals:   <bar>
terminals:      bar
Literals:       bar
Lists:          [list <of> items]
one or more:    +
zero or one:    ?
zero or more:   *
groups:         {  }
alternation:    |

<struct>    ::=  equation_specs = [<specs>  <specs>]

<specs>   ::=  unknowns = [SYMBOL+] | domains = [<domain>+]

<domain>  ::=  INT = [ regions = [<region>+] ]

<region>   ::=  { Interior
                | Top    | TopOL
                | Left   | LeftOL
                | Bottom | BottomOL
                | Right  | RightOL  } = [<equation>+]

<equation> ::=  <expr> ≐ <plain-expr>

<expr>     ::=  {<plain-expr> ⊗}? SYMBOL {\ INT}? {{⊕|⊖} <expr>}*

<plain-expr> ::=  {-|+}? <prod-expr> {{+|-} <plain-expr>}*

<prod-expr> ::=  <pow-expr> {{*|/} <prod-expr>}*

<pow-expr> ::=  <elem-expr> {{**|^} <pow-expr>}*

<elem-expr> ::=  FLOAT | INT | SYMBOL | <function-call> | <indexed-access>
```

Figure 4.3. A BNF grammar for the equation input format. The key to automatic input equation conversions are the <domain>, <region>, <equation> and <expr> rules. The rules for <plain-expr> are standard expression syntax.

unknown products (arguments to $\oplus$ or $\ominus$) is allowed. After this identification of operators, only identification of equation, region and domain remains. This is accomplished via the rules <equation>, <region> and <domain>, respectively.

The key to obtaining an exactly-specified structure is to make distinctions (1) between regular multiplication ($*$) and the multiplication of a coefficient and an unknown-operator term ($\otimes$), and (2) between regular addition (+) and addition of operator-unknown terms via $\oplus$. Also, the operators $\otimes$ and $\backslash$ are not commutative, further clarifying structure.

As an illustration, consider the Laplace equation of Chapter 2, with boundary conditions, written in the typeset version of this format:

$$
\begin{aligned}
&\mathsf{equation\_specs} = [\\
&\quad \mathsf{unknowns} = [\mathsf{u}_1, \mathsf{u}_{11}, \mathsf{u}_{12}],\\
&\quad \mathsf{domains} = [\\
&\qquad 1 = [\\
&\qquad\quad \mathsf{regions} = [\\
&\qquad\qquad \mathsf{Interior} = [\mathsf{u}_{11}\backslash_1 \oplus \mathsf{u}_{12}\backslash_2 \cong 0,\\
&\qquad\qquad\qquad \mathsf{u}_{11} \ominus \mathsf{u}_1\backslash_1 \cong 0, \mathsf{u}_{12} \ominus \mathsf{u}_1\backslash_2 \cong 0],\\
&\qquad\qquad \mathsf{Top} = [\mathsf{u}_1 \cong \mathsf{f}(\mathsf{x},\mathsf{y}),\\
&\qquad\qquad\qquad \mathsf{u}_{11} \ominus \mathsf{u}_1\backslash_1 \cong 0, \mathsf{u}_{12} \ominus \mathsf{u}_1\backslash_2 \cong 0],\\
&\qquad\qquad \mathsf{Left} = [\mathsf{u}_1 \cong \mathsf{f}(\mathsf{x},\mathsf{y}),\\
&\qquad\qquad\qquad \mathsf{u}_{11} \ominus \mathsf{u}_1\backslash_1 \cong 0, \mathsf{u}_{12} \ominus \mathsf{u}_1\backslash_2 \cong 0],\\
&\qquad\qquad \mathsf{Right} = [\mathsf{u}_1 \cong \mathsf{f}(\mathsf{x},\mathsf{y}),\\
&\qquad\qquad\qquad \mathsf{u}_{11} \ominus \mathsf{u}_1\backslash_1 \cong 0, \mathsf{u}_{12} \ominus \mathsf{u}_1\backslash_2 \cong 0],\\
&\qquad\qquad \mathsf{Bottom} = [\mathsf{u}_1\backslash_2 \cong 0,\\
&\qquad\qquad\qquad \mathsf{u}_{11} \ominus \mathsf{u}_1\backslash_1 \cong 0, \mathsf{u}_{12} \ominus \mathsf{u}_1\backslash_2 \cong 0]]]]]
\end{aligned}
$$

The three equations forming the content of the $\mathsf{Interior} = [\ldots]$ expression are derived from the second-order equation

$$\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} = 0 \tag{4.36}$$

by defining the unknowns u_{11} and u_{12} as

$$u_{11} = \partial u_1 / \partial x \tag{4.37}$$

$$u_{12} = \partial u_1 / \partial y. \tag{4.38}$$

and using these two equations along with the form

$$\partial u_{11} / \partial x + \partial u_{12} / \partial y = 0 \tag{4.39}$$

of the Laplace equation. The Equations 4.37 and 4.38 are also used in all other region specifications, along with the conditions for that region.

4.5.2 Block system

The individual blocks and the block system structure are directly derived from the BNF grammar in Figure 4.3. The matrix $[L]$ is block diagonal, except for blocks corresponding to overlapping regions. The number of diagonal blocks corresponds to the number of rectangles; every diagonal block's entries are column-aligned by unknown and row-aligned by the region and equation. The only special case is presented by the overlapping blocks; they are column-aligned with their own unknown and domain (just as regular blocks), and row-aligned with the matching boundary conditions' domain (as determined by its domains' layout relative to other domains) and region (as determined by the domains' layout and the region matching the block's region).

As an illustration, a single-unknown, single-rectangle, second-order elliptic PDE problem can be written in the form $Lu = f$ as

$$\begin{pmatrix} L^{1|1}_{1\wedge \mathsf{I}} \\ L^{1|1}_{1\wedge \mathsf{T}} \\ L^{1|1}_{1\wedge \mathsf{L}} \\ L^{1|1}_{1\wedge \mathsf{B}} \\ L^{1|1}_{1\wedge \mathsf{R}} \end{pmatrix} u^{1|1} = \begin{pmatrix} g^{|1}_{1\wedge \mathsf{I}} \\ g^{|1}_{1\wedge \mathsf{T}} \\ g^{|1}_{1\wedge \mathsf{L}} \\ g^{|1}_{1\wedge \mathsf{B}} \\ g^{|1}_{1\wedge \mathsf{R}} \end{pmatrix} \tag{4.40}$$

By introducing the new unknowns $u^{11|1}$ and $u^{12|1}$ and the equations

$$u^{11|1} = \partial u^{1|1}/\partial x \tag{4.41}$$

$$u^{12|1} = \partial u^{1|1}/\partial y \tag{4.42}$$

all occurrences of second-order partials can be replaced with first-order expressions; omitting blocks of zeroes, the equivalent first-order system then has the form

$$\begin{pmatrix}
\overline{L}^{1|1}_{1\wedge\mathsf{I}} & \overline{L}^{11|1}_{1\wedge\mathsf{I}} & \overline{L}^{12|1}_{1\wedge\mathsf{I}} \\
\overline{L}^{1|1}_{2\wedge\mathsf{I}} & \overline{L}^{11|1}_{2\wedge\mathsf{I}} & \\
\overline{L}^{1|1}_{3\wedge\mathsf{I}} & & \overline{L}^{12|1}_{3\wedge\mathsf{I}} \\
\overline{L}^{1|1}_{1\wedge\mathsf{T}} & \overline{L}^{11|1}_{1\wedge\mathsf{T}} & \overline{L}^{12|1}_{1\wedge\mathsf{T}} \\
\overline{L}^{1|1}_{2\wedge\mathsf{T}} & \overline{L}^{11|1}_{2\wedge\mathsf{T}} & \\
\overline{L}^{1|1}_{3\wedge\mathsf{T}} & & \overline{L}^{12|1}_{3\wedge\mathsf{T}} \\
\overline{L}^{1|1}_{1\wedge\mathsf{L}} & \overline{L}^{11|1}_{1\wedge\mathsf{L}} & \overline{L}^{12|1}_{1\wedge\mathsf{L}} \\
\overline{L}^{1|1}_{2\wedge\mathsf{L}} & \overline{L}^{11|1}_{2\wedge\mathsf{L}} & \\
\overline{L}^{1|1}_{3\wedge\mathsf{L}} & & \overline{L}^{12|1}_{3\wedge\mathsf{L}} \\
\overline{L}^{1|1}_{1\wedge\mathsf{B}} & \overline{L}^{11|1}_{1\wedge\mathsf{B}} & \overline{L}^{12|1}_{1\wedge\mathsf{B}} \\
\overline{L}^{1|1}_{2\wedge\mathsf{B}} & \overline{L}^{11|1}_{2\wedge\mathsf{B}} & \\
\overline{L}^{1|1}_{3\wedge\mathsf{B}} & & \overline{L}^{12|1}_{3\wedge\mathsf{B}} \\
\overline{L}^{1|1}_{1\wedge\mathsf{R}} & \overline{L}^{11|1}_{1\wedge\mathsf{R}} & \overline{L}^{12|1}_{1\wedge\mathsf{R}} \\
\overline{L}^{1|1}_{2\wedge\mathsf{R}} & \overline{L}^{11|1}_{2\wedge\mathsf{R}} & \\
\overline{L}^{1|1}_{3\wedge\mathsf{R}} & & \overline{L}^{12|1}_{3\wedge\mathsf{R}}
\end{pmatrix}
\begin{pmatrix} u^{1|1} \\ u^{11|1} \\ u^{12|1} \end{pmatrix}
=
\begin{pmatrix}
\overline{g}^{|1}_{1\wedge\mathsf{I}} \\ \\ \\ \overline{g}^{|1}_{1\wedge\mathsf{T}} \\ \\ \\ \overline{g}^{|1}_{1\wedge\mathsf{L}} \\ \\ \\ \overline{g}^{|1}_{1\wedge\mathsf{B}} \\ \\ \\ \overline{g}^{|1}_{1\wedge\mathsf{R}} \\ \\ \\
\end{pmatrix} \tag{4.43}$$

or

$$\overline{L}\overline{u} = \overline{f} \tag{4.44}$$

4.5.3 Discrete block system

The discrete block system structure is visually identical to that of the block system; the differences in the blocks come from the discretization, which introduces the unknowns' coefficients and the regions' collocation points. Full details on the ordering of these new parts are not relevant here, and we can proceed in a general manner as follows. Let $k \in [-N, N], l \in [-N, N]$. Define $m \equiv 2N+1$, and let $j \in [0, m^2-1]$. Define a discrete one-to-one mapping

$$\tau \ : (k, l) \to j.$$

In the following, let i be an enumeration of all collocation points of the current appropriate region, and j an enumeration of all unknown terms of the current appropriate unknown. Define

$$k_j, l_j \ = \tau^{-1}(j)$$

and form the following discrete matrix blocks:

$$\begin{aligned} \left[\mathsf{L}^{1|1}_{1\wedge\mathsf{I}}\right]_{ij} &= \left(L^{1|1}_{1\wedge\mathsf{I}}\right)(\omega^{|1,x}_{k_j}\omega^{|1,y}_{l_j})(x_i, y_i), \\ &\quad (x_i, y_i) \ \in \mathsf{I} \end{aligned} \tag{4.45}$$

$$\begin{aligned} \left[\mathsf{L}^{1|1}_{1\wedge\mathsf{T}}\right]_{ij} &= \left(L^{1|1}_{1\wedge\mathsf{T}}\right)(\omega^{|1,x}_{k_j}\omega^{|1,y}_{l_j})(x_i, y_i), \\ &\quad (x_i, y_i) \ \in \mathsf{T} \end{aligned} \tag{4.46}$$

$$\begin{aligned} \left[\mathsf{L}^{1|1}_{1\wedge\mathsf{L}}\right]_{ij} &= \left(L^{1|1}_{1\wedge\mathsf{L}}\right)(\omega^{|1,x}_{k_j}\omega^{|1,y}_{l_j})(x_i, y_i), \\ &\quad (x_i, y_i) \ \in \mathsf{L} \end{aligned} \tag{4.47}$$

$$\begin{aligned} \left[\mathsf{L}^{1|1}_{1\wedge\mathsf{B}}\right]_{ij} &= \left(L^{1|1}_{1\wedge\mathsf{B}}\right)(\omega^{|1,x}_{k_j}\omega^{|1,y}_{l_j})(x_i, y_i), \\ &\quad (x_i, y_i) \ \in \mathsf{B} \end{aligned} \tag{4.48}$$

$$\begin{aligned} \left[\mathsf{L}^{1|1}_{1\wedge\mathsf{R}}\right]_{ij} &= \left(L^{1|1}_{1\wedge\mathsf{R}}\right)(\omega^{|1,x}_{k_j}\omega^{|1,y}_{l_j})(x_i, y_i), \\ &\quad (x_i, y_i) \ \in \mathsf{R} \end{aligned} \tag{4.49}$$

$$[u^{1|1}]_j \quad = \quad c^{1|1}_{k_j l_j} \tag{4.50}$$

$$[g^{|1}_{1\wedge m}]_i \quad = \quad g^{|1}_{1\wedge m}(x_i, y_i) \tag{4.51}$$

$$m \ \in [\mathsf{I}, \mathsf{T}, \mathsf{L}, \mathsf{B}, \mathsf{R}]$$

Then the operator form in Equation 4.40 has the discrete block analogue

$$\begin{pmatrix} \left[\mathrm{L}_{1\wedge\mathsf{I}}^{1|1}\right] \\ \left[\mathrm{L}_{1\wedge\mathsf{T}}^{1|1}\right] \\ \left[\mathrm{L}_{1\wedge\mathsf{L}}^{1|1}\right] \\ \left[\mathrm{L}_{1\wedge\mathsf{B}}^{1|1}\right] \\ \left[\mathrm{L}_{1\wedge\mathsf{R}}^{1|1}\right] \end{pmatrix} \left[u^{1|1}\right] = \begin{pmatrix} \left[g_{1\wedge\mathsf{I}}^{|1}\right] \\ \left[g_{1\wedge\mathsf{T}}^{|1}\right] \\ \left[g_{1\wedge\mathsf{L}}^{|1}\right] \\ \left[g_{1\wedge\mathsf{B}}^{|1}\right] \\ \left[g_{1\wedge\mathsf{R}}^{|1}\right] \end{pmatrix} \tag{4.52}$$

or

$$[L][u] = [f] \tag{4.53}$$

Note that while none of the constituent matrix blocks is square, the matrix $[L]$ is.

By forming discrete matrix blocks for the system of Equation 4.43 in the same manner, the operator form of Equation 4.43 has the discrete block analogue

$$\begin{pmatrix} [\overline{L}_{1\wedge\mathsf{I}}^{1|1}] & [\overline{L}_{1\wedge\mathsf{I}}^{11|1}] & [\overline{L}_{1\wedge\mathsf{I}}^{12|1}] \\ [\overline{L}_{2\wedge\mathsf{I}}^{1|1}] & [\overline{L}_{2\wedge\mathsf{I}}^{11|1}] & \\ [\overline{L}_{3\wedge\mathsf{I}}^{1|1}] & & [\overline{L}_{3\wedge\mathsf{I}}^{12|1}] \\ [\overline{L}_{1\wedge\mathsf{T}}^{1|1}] & [\overline{L}_{1\wedge\mathsf{T}}^{11|1}] & [\overline{L}_{1\wedge\mathsf{T}}^{12|1}] \\ [\overline{L}_{2\wedge\mathsf{T}}^{1|1}] & [\overline{L}_{2\wedge\mathsf{T}}^{11|1}] & \\ [\overline{L}_{3\wedge\mathsf{T}}^{1|1}] & & [\overline{L}_{3\wedge\mathsf{T}}^{12|1}] \\ [\overline{L}_{1\wedge\mathsf{L}}^{1|1}] & [\overline{L}_{1\wedge\mathsf{L}}^{11|1}] & [\overline{L}_{1\wedge\mathsf{L}}^{12|1}] \\ [\overline{L}_{2\wedge\mathsf{L}}^{1|1}] & [\overline{L}_{2\wedge\mathsf{L}}^{11|1}] & \\ [\overline{L}_{3\wedge\mathsf{L}}^{1|1}] & & [\overline{L}_{3\wedge\mathsf{L}}^{12|1}] \\ [\overline{L}_{1\wedge\mathsf{B}}^{1|1}] & [\overline{L}_{1\wedge\mathsf{B}}^{11|1}] & [\overline{L}_{1\wedge\mathsf{B}}^{12|1}] \\ [\overline{L}_{2\wedge\mathsf{B}}^{1|1}] & [\overline{L}_{2\wedge\mathsf{B}}^{11|1}] & \\ [\overline{L}_{3\wedge\mathsf{B}}^{1|1}] & & [\overline{L}_{3\wedge\mathsf{B}}^{12|1}] \\ [\overline{L}_{1\wedge\mathsf{R}}^{1|1}] & [\overline{L}_{1\wedge\mathsf{R}}^{11|1}] & [\overline{L}_{1\wedge\mathsf{R}}^{12|1}] \\ [\overline{L}_{2\wedge\mathsf{R}}^{1|1}] & [\overline{L}_{2\wedge\mathsf{R}}^{11|1}] & \\ [\overline{L}_{3\wedge\mathsf{R}}^{1|1}] & & [\overline{L}_{3\wedge\mathsf{R}}^{12|1}] \end{pmatrix} \begin{pmatrix} [u^{1|1}] \\ [u^{11|1}] \\ [u^{12|1}] \end{pmatrix} = \begin{pmatrix} [\overline{g}_{1\wedge\mathsf{I}}^{|1}] \\ \\ \\ [\overline{g}_{1\wedge\mathsf{T}}^{|1}] \\ \\ \\ [\overline{g}_{1\wedge\mathsf{L}}^{|1}] \\ \\ \\ [\overline{g}_{1\wedge\mathsf{B}}^{|1}] \\ \\ \\ [\overline{g}_{1\wedge\mathsf{R}}^{|1}] \\ \\ \\ \end{pmatrix} \tag{4.54}$$

or

$$[\overline{L}][\overline{u}] = [\overline{f}] \tag{4.55}$$

To show the overall error bound for the discretization $[\overline{L}][\overline{u}]$ is $O(\epsilon_N)$, we start with the error bounds on individual matrix blocks obtained from Theorem 4.7, expand the matrix-vector products, and arrive at the difference $[\overline{Lu}] - [\overline{L}][\overline{u}]$.

For every block, an $O(\epsilon_N)$ error is incurred in the discretized version of the block system. This follows directly from the observation that for all (x, y) in a given rectangle, Theorem 4.7 allow us to write

$$[\overline{L}^{i|l}_{j\wedge k} u^{i|j}] \; = [\overline{L}^{i|l}_{j\wedge k}][u^{i|j}] + [\epsilon^{i|j}_N] \tag{4.56}$$

for every i, j, k and l; all $[\epsilon^{i|j}_N]$ are $O(Ne^{-d\sqrt{N}})$ for derivatives, or $O(\sqrt{N}e^{-d\sqrt{N}})$ for function values, e.g., for the upper-left block of Equation 4.43,

$$[\overline{L}^{1|1}_{1\wedge \mathsf{I}} u^{1|1}] \; = [\overline{L}^{1|1}_{1\wedge \mathsf{I}}][u^{1|1}] + [\epsilon^{1|1}_N]. \tag{4.57}$$

Continuing the abstract sample problem and using Equation 4.43, we can write

$$[\overline{Lu}] \; = \begin{pmatrix} [\overline{L}^{1|1}_{1\wedge \mathsf{I}} u^{1|1} + \overline{L}^{11|1}_{1\wedge \mathsf{I}} u^{11|1} + \overline{L}^{12|1}_{1\wedge \mathsf{I}} u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge \mathsf{I}} u^{1|1} + \overline{L}^{11|1}_{2\wedge \mathsf{I}} u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge \mathsf{I}} u^{1|1} + \overline{L}^{12|1}_{3\wedge \mathsf{I}} u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge \mathsf{T}} u^{1|1} + \overline{L}^{11|1}_{1\wedge \mathsf{T}} u^{11|1} + \overline{L}^{12|1}_{1\wedge \mathsf{T}} u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge \mathsf{T}} u^{1|1} + \overline{L}^{11|1}_{2\wedge \mathsf{T}} u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge \mathsf{T}} u^{1|1} + \overline{L}^{12|1}_{3\wedge \mathsf{T}} u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge \mathsf{L}} u^{1|1} + \overline{L}^{11|1}_{1\wedge \mathsf{L}} u^{11|1} + \overline{L}^{12|1}_{1\wedge \mathsf{L}} u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge \mathsf{L}} u^{1|1} + \overline{L}^{11|1}_{2\wedge \mathsf{L}} u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge \mathsf{L}} u^{1|1} + \overline{L}^{12|1}_{3\wedge \mathsf{L}} u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge \mathsf{B}} u^{1|1} + \overline{L}^{11|1}_{1\wedge \mathsf{B}} u^{11|1} + \overline{L}^{12|1}_{1\wedge \mathsf{B}} u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge \mathsf{B}} u^{1|1} + \overline{L}^{11|1}_{2\wedge \mathsf{B}} u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge \mathsf{B}} u^{1|1} + \overline{L}^{12|1}_{3\wedge \mathsf{B}} u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge \mathsf{R}} u^{1|1} + \overline{L}^{11|1}_{1\wedge \mathsf{R}} u^{11|1} + \overline{L}^{12|1}_{1\wedge \mathsf{R}} u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge \mathsf{R}} u^{1|1} + \overline{L}^{11|1}_{2\wedge \mathsf{R}} u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge \mathsf{R}} u^{1|1} + \overline{L}^{12|1}_{3\wedge \mathsf{R}} u^{12|1}] \end{pmatrix} = [\overline{L}][\overline{u}] + [\overline{\epsilon}_N] \tag{4.58}$$

where

$$[\overline{L}][\overline{u}] = \begin{pmatrix} [\overline{L}^{1|1}_{1\wedge\mathsf{I}}][u^{1|1}] + [\overline{L}^{11|1}_{1\wedge\mathsf{I}}][u^{11|1}] + [\overline{L}^{12|1}_{1\wedge\mathsf{I}}][u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge\mathsf{I}}][u^{1|1}] + [\overline{L}^{11|1}_{2\wedge\mathsf{I}}][u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge\mathsf{I}}][u^{1|1}] + [\overline{L}^{12|1}_{3\wedge\mathsf{I}}][u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge\mathsf{T}}][u^{1|1}] + [\overline{L}^{11|1}_{1\wedge\mathsf{T}}][u^{11|1}] + [\overline{L}^{12|1}_{1\wedge\mathsf{T}}][u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge\mathsf{T}}][u^{1|1}] + [\overline{L}^{11|1}_{2\wedge\mathsf{T}}][u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge\mathsf{T}}][u^{1|1}] + [\overline{L}^{12|1}_{3\wedge\mathsf{T}}][u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge\mathsf{L}}][u^{1|1}] + [\overline{L}^{11|1}_{1\wedge\mathsf{L}}][u^{11|1}] + [\overline{L}^{12|1}_{1\wedge\mathsf{L}}][u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge\mathsf{L}}][u^{1|1}] + [\overline{L}^{11|1}_{2\wedge\mathsf{L}}][u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge\mathsf{L}}][u^{1|1}] + [\overline{L}^{12|1}_{3\wedge\mathsf{L}}][u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge\mathsf{B}}][u^{1|1}] + [\overline{L}^{11|1}_{1\wedge\mathsf{B}}][u^{11|1}] + [\overline{L}^{12|1}_{1\wedge\mathsf{B}}][u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge\mathsf{B}}][u^{1|1}] + [\overline{L}^{11|1}_{2\wedge\mathsf{B}}][u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge\mathsf{B}}][u^{1|1}] + [\overline{L}^{12|1}_{3\wedge\mathsf{B}}][u^{12|1}] \\ [\overline{L}^{1|1}_{1\wedge\mathsf{R}}][u^{1|1}] + [\overline{L}^{11|1}_{1\wedge\mathsf{R}}][u^{11|1}] + [\overline{L}^{12|1}_{1\wedge\mathsf{R}}][u^{12|1}] \\ [\overline{L}^{1|1}_{2\wedge\mathsf{R}}][u^{1|1}] + [\overline{L}^{11|1}_{2\wedge\mathsf{R}}][u^{11|1}] \\ [\overline{L}^{1|1}_{3\wedge\mathsf{R}}][u^{1|1}] + [\overline{L}^{12|1}_{3\wedge\mathsf{R}}][u^{12|1}] \end{pmatrix}, \tag{4.59}$$

and

$$[\overline{\epsilon}_N] = \begin{pmatrix} [\epsilon^{1|1}_{1\wedge\mathsf{I}}] + [\epsilon^{11|1}_{1\wedge\mathsf{I}}] + [\epsilon^{12|1}_{1\wedge\mathsf{I}}] \\ [\epsilon^{1|1}_{2\wedge\mathsf{I}}] + [\epsilon^{11|1}_{2\wedge\mathsf{I}}] \\ [\epsilon^{1|1}_{3\wedge\mathsf{I}}] + [\epsilon^{12|1}_{3\wedge\mathsf{I}}] \\ [\epsilon^{1|1}_{1\wedge\mathsf{T}}] + [\epsilon^{11|1}_{1\wedge\mathsf{T}}] + [\epsilon^{12|1}_{1\wedge\mathsf{T}}] \\ [\epsilon^{1|1}_{2\wedge\mathsf{T}}] + [\epsilon^{11|1}_{2\wedge\mathsf{T}}] \\ [\epsilon^{1|1}_{3\wedge\mathsf{T}}] + [\epsilon^{12|1}_{3\wedge\mathsf{T}}] \\ [\epsilon^{1|1}_{1\wedge\mathsf{L}}] + [\epsilon^{11|1}_{1\wedge\mathsf{L}}] + [\epsilon^{12|1}_{1\wedge\mathsf{L}}] \\ [\epsilon^{1|1}_{2\wedge\mathsf{L}}] + [\epsilon^{11|1}_{2\wedge\mathsf{L}}] \\ [\epsilon^{1|1}_{3\wedge\mathsf{L}}] + [\epsilon^{12|1}_{3\wedge\mathsf{L}}] \\ [\epsilon^{1|1}_{1\wedge\mathsf{B}}] + [\epsilon^{11|1}_{1\wedge\mathsf{B}}] + [\epsilon^{12|1}_{1\wedge\mathsf{B}}] \\ [\epsilon^{1|1}_{2\wedge\mathsf{B}}] + [\epsilon^{11|1}_{2\wedge\mathsf{B}}] \\ [\epsilon^{1|1}_{3\wedge\mathsf{B}}] + [\epsilon^{12|1}_{3\wedge\mathsf{B}}] \\ [\epsilon^{1|1}_{1\wedge\mathsf{R}}] + [\epsilon^{11|1}_{1\wedge\mathsf{R}}] + [\epsilon^{12|1}_{1\wedge\mathsf{R}}] \\ [\epsilon^{1|1}_{2\wedge\mathsf{R}}] + [\epsilon^{11|1}_{2\wedge\mathsf{R}}] \\ [\epsilon^{1|1}_{3\wedge\mathsf{R}}] + [\epsilon^{12|1}_{3\wedge\mathsf{R}}] \end{pmatrix}, \tag{4.60}$$

so the overall discretization error $[\overline{\epsilon}_N]$ remains $O(Ne^{-d\sqrt{N}})$.

4.5.4 Discrete approximation

In the following, it is assumed that the PDE system is well-posed, so the resulting discrete block system is uniquely invertible and numerically nonsingular. Under these assumptions, any stable solution method then obtains a vector $[c]$ satisfying

$$[\overline{L}][c] \; = [\overline{f}] + [\overline{\epsilon}_u] \tag{4.61}$$

with $[\overline{\epsilon}_u]$ proportional to the unit roundoff error.

4.5.5 Smooth approximation

Following the steps of Section 4.3, we see that

$$\|[\overline{u}] - [c]\| \; \leq \|[\overline{L}]^{-1}\| \left\{ \left\|[\overline{\epsilon}_N]\right\| + \left\|[\overline{\epsilon}_u]\right\| \right\} \tag{4.62}$$

and Theorem 4.8 therefore applies. The coefficients $[u^{i|j}]_k$ can thus be extracted from $[c]$ and used to obtain $u^{i|j}(x, y)$ for any $(x, y) \in \Omega_j$ via Equation 4.32; by Theorem 4.8, this introduces the same $O(\epsilon_N)$ error as the discretization steps.

CHAPTER 5

NUMERICAL RESULTS

In this chapter, we explore the suitability of the SINC-ELLPDE method for singular problems via the numerical solution of the problems of Sections 3.1 and 3.2. This will illustrate the numerical convergence behavior of the method in the presence of multiple domains and unknowns, as well as providing insight into the pointwise behavior of derivatives under these conditions.

5.1 Single-material crack

Presented here are the numerical results obtained for the problem of Section 3.1, using the exact answer given by (Sneddon 1995) for comparisons in appropriate regions (i.e., those near the crack). The full equations solved here are those shown graphically in Figure 3.2, which in turn use the expansions in Equations 3.13 and 3.14 to result in the following equations, written in the PDE system format of Section 4.5.1, with common system equations factored for clarity.

$$
\begin{array}{l}
\mathsf{equation\_specs} = [\\
\quad \mathsf{unknowns} = [\, \mathsf{u}_1, \quad \mathsf{u}_2, \quad \mathsf{u}_{11}, \quad \mathsf{u}_{12}, \quad \mathsf{u}_{21}, \quad \mathsf{u}_{22}],\\
\quad \mathsf{domains} = [\\
\qquad 1 = [\\
\qquad\quad \mathsf{regions} = [\\
\qquad\qquad \mathsf{Interior} = [\\
\qquad\qquad\quad (((-2)\nu + 2) \odot \mathsf{u}_{11}\backslash_1 \oplus (1-2\nu) \odot \mathsf{u}_{12}\backslash_2) \oplus\\
\qquad\qquad\qquad 1 \odot \mathsf{u}_{21}\backslash_2 \cong 0\ ,\\
\qquad\qquad\quad ((1-2\nu) \odot \mathsf{u}_{21}\backslash_1 \oplus ((-2)\nu + 2) \odot \mathsf{u}_{22}\backslash_2) \oplus\\
\qquad\qquad\qquad 1 \odot \mathsf{u}_{11}\backslash_2 \cong 0\ ,\\
\qquad\qquad\quad \mathsf{system}],\\
\qquad\qquad \mathsf{Bottom} = [\\
\qquad\qquad\quad -\mu \odot \mathsf{u}_1\backslash_2 \oplus -\mu \odot \mathsf{u}_2\backslash_1 \cong 0\ ,
\end{array}
$$

$$\frac{(-2)\mu((-1)+\nu)}{(-1)+2\nu} \odot \mathsf{u}_2\backslash_2 \oplus$$
$$\frac{2\mu\nu}{(-1)+2\nu} \odot \mathsf{u}_1\backslash_1 \cong \sigma\,,$$

```
                system],
        Left= [ 1 ⊙ u21 ≅ 0 ,   1 ⊙ u1 ≅ 0 ,   system],
        Top= [ 1 ⊙ u1 ≅ 0 ,   1 ⊙ u2 ≅ 0 ,   system],
        Right= [ 1 ⊙ u11 ≅ 0 ,   1 ⊙ u21 ≅ 0 ,   system],
        RightOL= [ 1 ⊙ u1 ≅ 0 ,   1 ⊙ u2 ≅ 0 ]]],
  2 = [
     regions= [
        LeftOL= [ (−1) ⊙ u11 ≅ 0 ,   (−1) ⊙ u21 ≅ 0 ],
        Left= [ (−1) ⊙ u1 ≅ 0 ,   (−1) ⊙ u2 ≅ 0 ,
system],
        Top= [ 1 ⊙ u1 ≅ 0 ,   1 ⊙ u2 ≅ 0 ,   system],
        Right= [ 1 ⊙ u1 ≅ 0 ,   1 ⊙ u2 ≅ 0 ,   system],
        Interior= [
           (((−2)ν + 2) ⊙ u11\1 ⊕ (1 − 2ν) ⊙ u12\2)⊕
              1 ⊙ u21\2 ≅ 0 ,
           ((1 − 2ν) ⊙ u21\1 ⊕ ((−2)ν + 2) ⊙ u22\2)⊕
              1 ⊙ u11\2 ≅ 0 ,
           system],
        Bottom= [ 1 ⊙ u12 ≅ 0 ,   1 ⊙ u2 ≅ 0 ,   system]]]]]
system= [
   1 ⊙ u1\1 ⊕ (−1) ⊙ u11 ≅ 0 ,
   1 ⊙ u1\2 ⊕ (−1) ⊙ u12 ≅ 0 ,
   1 ⊙ u2\1 ⊕ (−1) ⊙ u21 ≅ 0 ,
   1 ⊙ u2\2 ⊕ (−1) ⊙ u22 ≅ 0 ]
```

The numerical parameters are shown in Table 5.1; the general geometry was shown in Figure 3.1 and is shown drawn to scale along with the individual graphs. As in Chapter 2, two approaches are used to illustrate convergence: a normwise error, and a pointwise error.

Table 5.1. Single-material problem parameters. σ is the applied crack load, ν is Poisson's ratio, and μ is one of the Lamé constants.

Geometry		Material Constants		Sinc Constants	
c	1.0	σ	1.0	α	1.0
w	27.0	ν	0.33	d	$\pi/2$
h	27.0	μ	2.0		

5.1.1 Convergence in norm

Since both stresses and displacements have zero exact value at several points of the elastic body, the relative error $|(e-a)/e|$ cannot be used effectively as a measure of error. Further, stresses are unbounded near the crack tip. This precludes use of the L^∞ norm of the absolute error $|e-a|$ as the absolute error also diverges and is unrealistically large near the singularities. However, using any p-norm of the absolute error, $1 \le p < \infty$, will work.

For the first convergence check, we show simple graphs of the number of terms N vs. the scaled norm of the absolute error,

$$\|f\| = \frac{\int_\Omega |f(x)| dx}{\int_\Omega dx}, \tag{5.1}$$

over several regions Ω.

These graphs are split across two figures. Figure 5.1 is split into three parts. The upper region shows the boundaries of the computational rectangles as solid lines; the two regions of integration (Ω_1 and Ω_2) are shown by the dashed lines. The crack is located at $y = 0$, $-1 \le x \le 1$. The lower left region displays convergence plots for the unknowns of rectangle 1 over Ω_1, while the lower right region displays convergence plots for the unknowns of rectangle 2 over Ω_2.

Figure 5.2 shows the convergence graphs for the individual stress components. Again, the left region displays results over Ω_1 while the right region displays results over Ω_2. All convergence plots show the computed error norms as triangular data points and the curve fit of the sinc convergence rate $ce^{-g\sqrt{N}}$ as a solid line.

Unlike the simple sample problem of Chapter 2, the measured error does not fit the theoretical bounds very well; it is erratic. A fit of the theoretical convergence rate (using values of $N \ge 8$) yields a rate constant g ranging from 2.4 to 2.9, higher than expected. Even so, the convergence is clearly still exponential, and good solutions are expected for larger N.

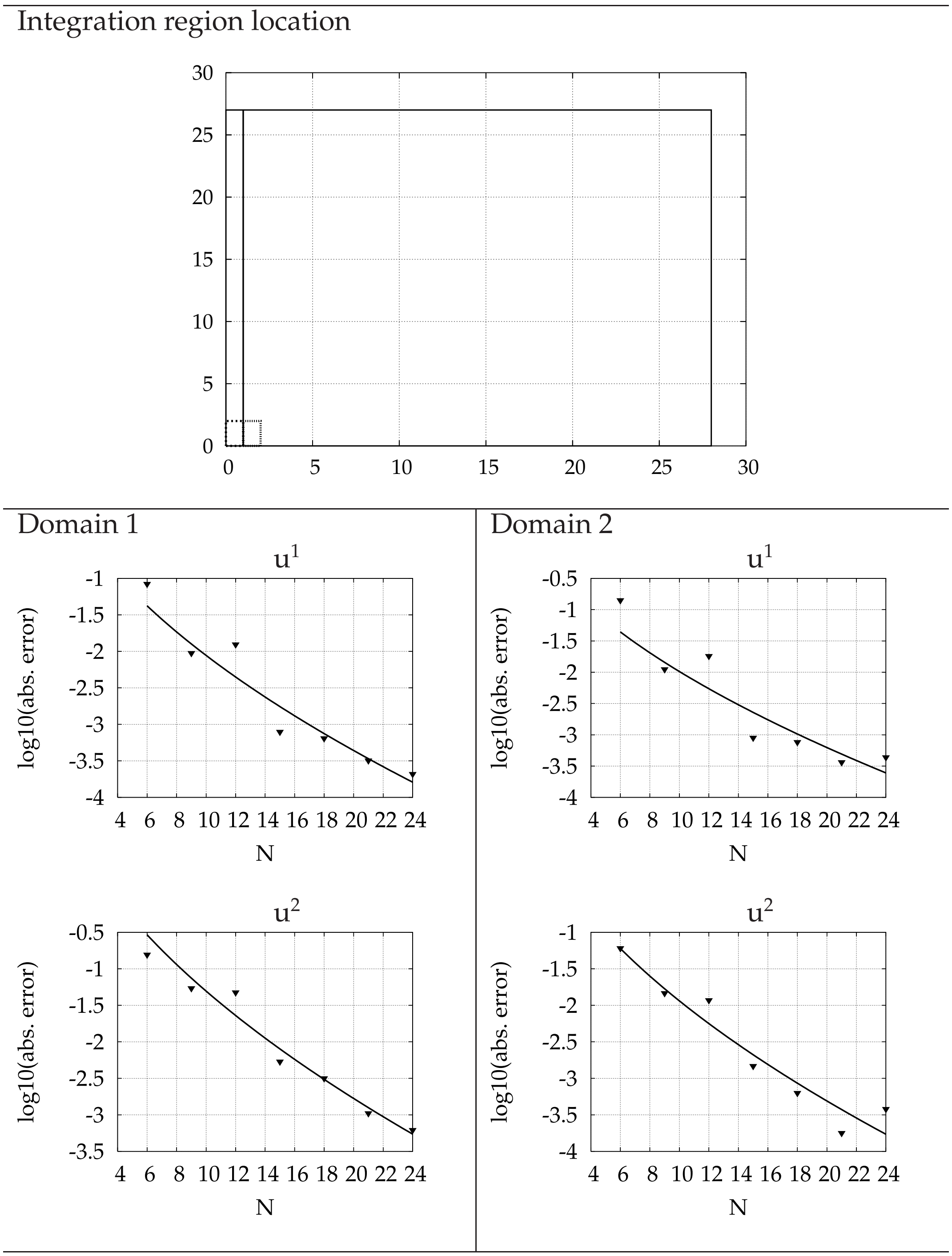


Figure 5.1. Graphs of L^1 convergence, part 1: Location of integration region and displacement error norms.

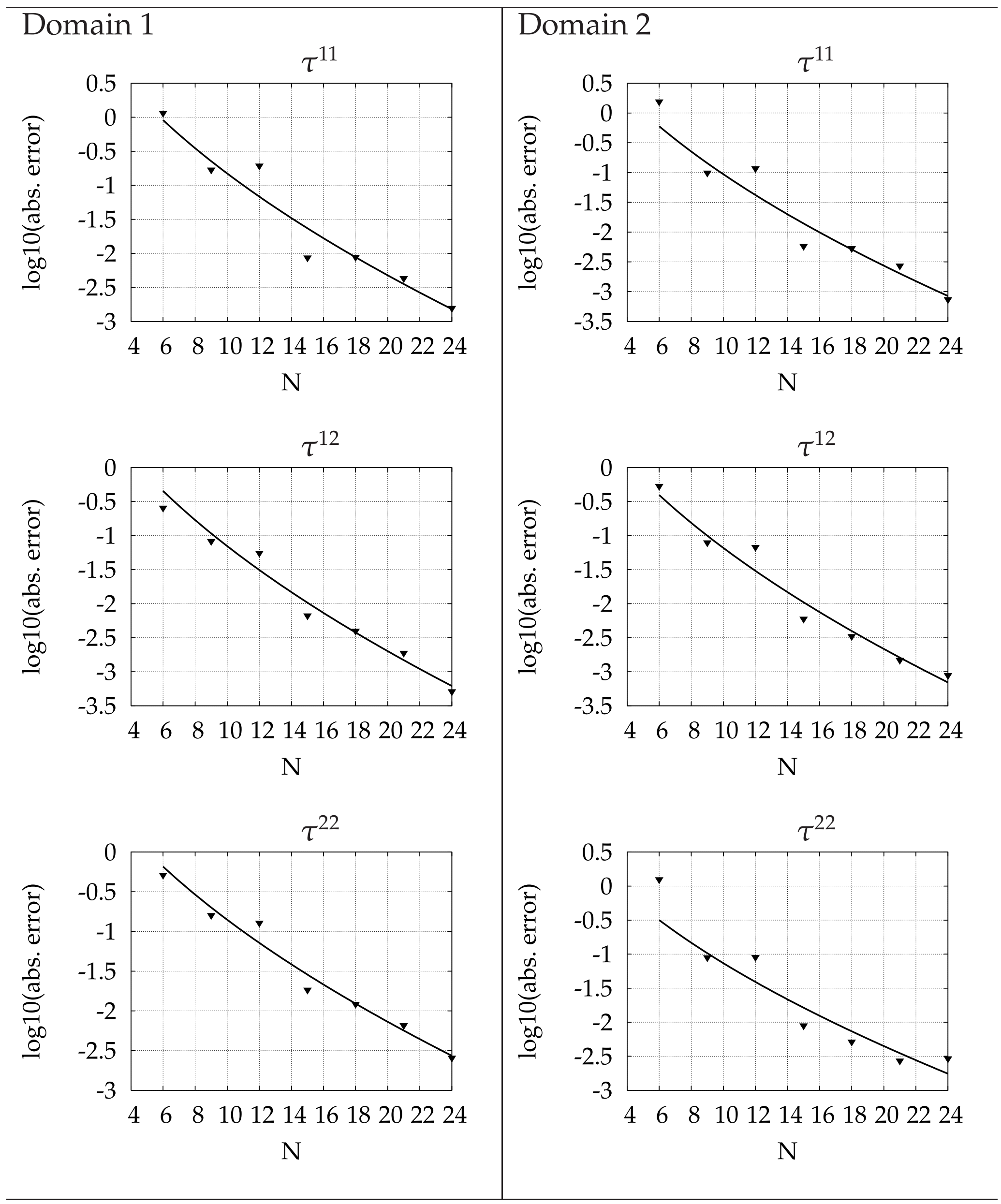


Figure 5.2. Graphs of L^1 convergence, part 2: Stress error norms.

5.1.2 Pointwise convergence

The normwise convergence checks in Section 5.1.1 give a global indication of convergence, but say nothing about the local quality of approximation.

To provide some insight, the following graphs illustrate the pointwise convergence behavior of the sinc method, one unknown at a time, over selected regions Ω_i near the crack tip.

A combination of three-dimensional surface- and two-dimensional xy-plots are shown; the surfaces provide both a qualitative overview and a reference frame, while the xy-plots provide quantitative data along slices.

The graphs for every unknown are partitioned into two sets of figures. The first set of figures displays the current data's location (on the "Area location" graph, via a solid rectangle), the legends for the xy-slices ("slice legend" table) and a surface view of the data ("Area surface view" graph). The surface view shows the surface formed by the data; lines on the surface and their projections onto the base show the location of the xy-slices. The base projections are numbered for cross-reference with the xy-slices' graphs. The second and third sets of figures show the detail slices' graphs. Each slice is numbered according to its position on the surface view graph.

The unknowns u^1, u^2 and the stresses τ^{11}, τ^{12} and τ^{22} are shown, each on two areas, in Figures 5.3 through 5.26.

There are two reasons for the present selection of viewing regions.

First, this problem is similar, but not identical, to the one solved by (Sneddon 1995); in particular, the solutions are only comparable in regions near the crack tip, and therefore the area-surface views focus on those regions.

Second, the sinc basis functions have unbounded derivatives at the rectangle boundaries; for the simple problem of Chapter 2, accurate derivative approximations were easily obtained on the range $(0.001, 0.99)$, out of $(0, 1)$. The present problem has (1) a more singular solution, (2) more unknowns, (3) more rectangles, and (4) a low number of terms m. These factors combine to yield very poor derivative approximations near boundaries; this per-domain gap is

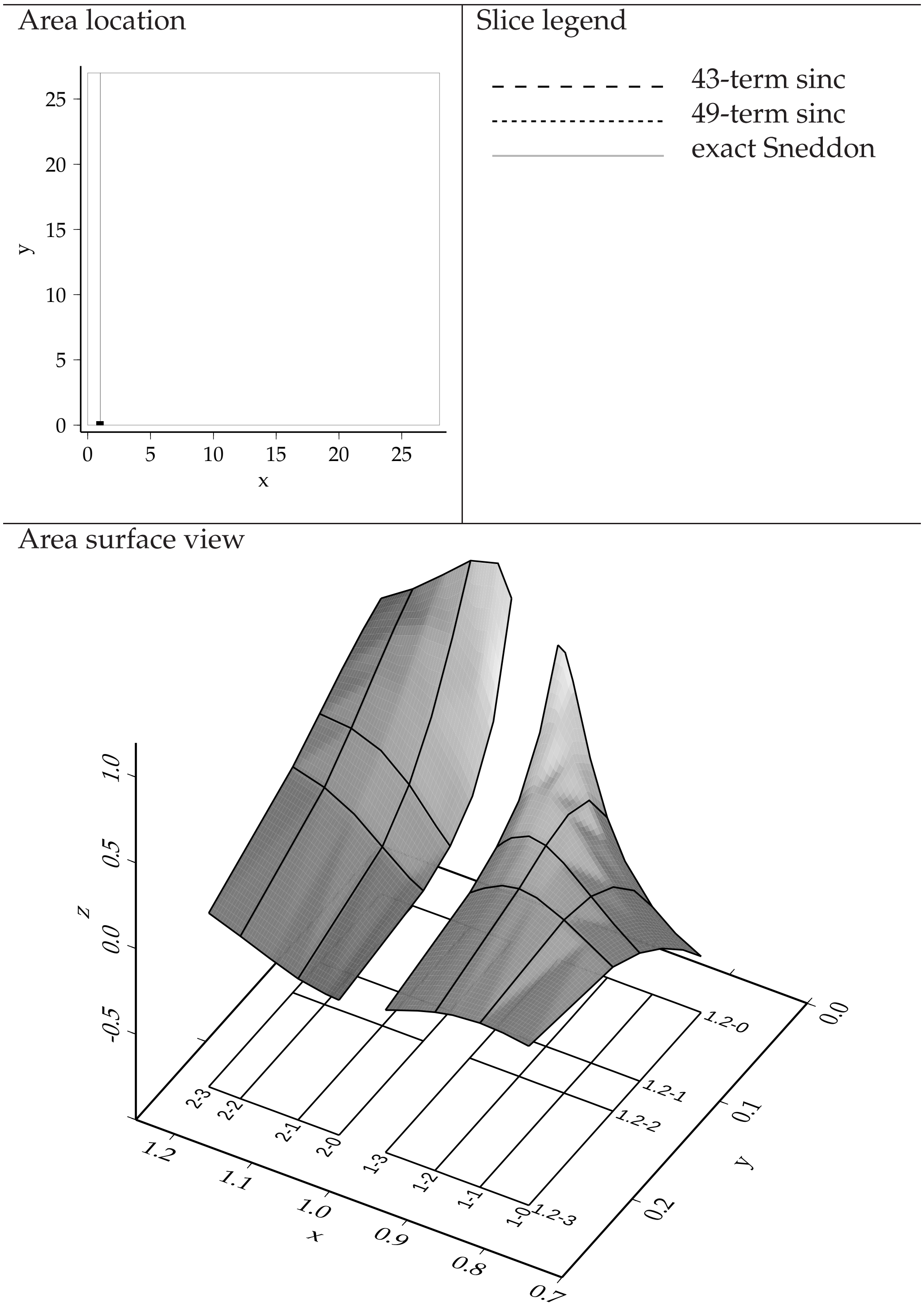


Figure 5.3. Graphs of τ^{11}, view 1, part 1: Location, legends and surface.

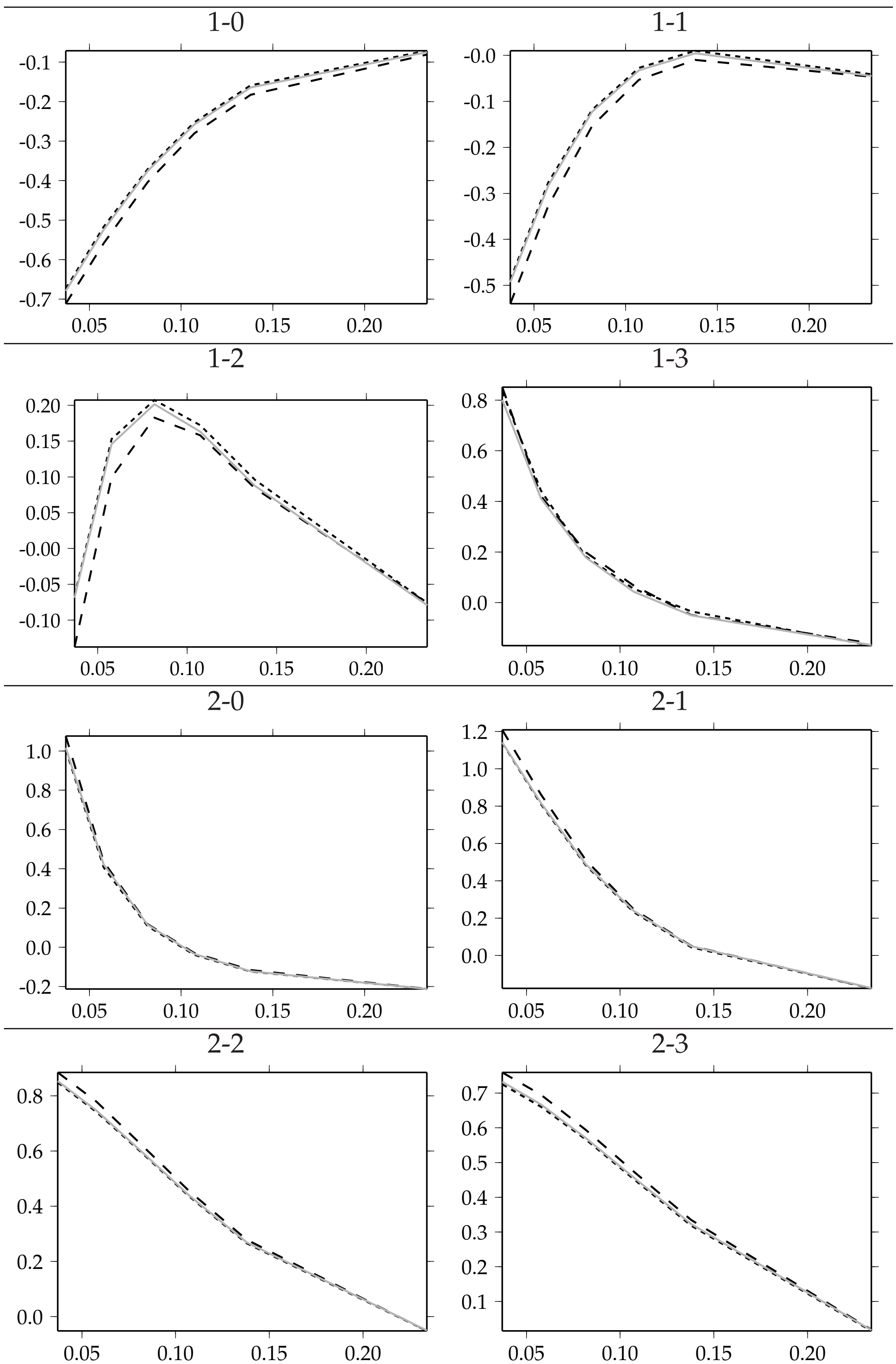


Figure 5.4. Graphs of τ^{11}, view 1, part 2: Detailed single slices.

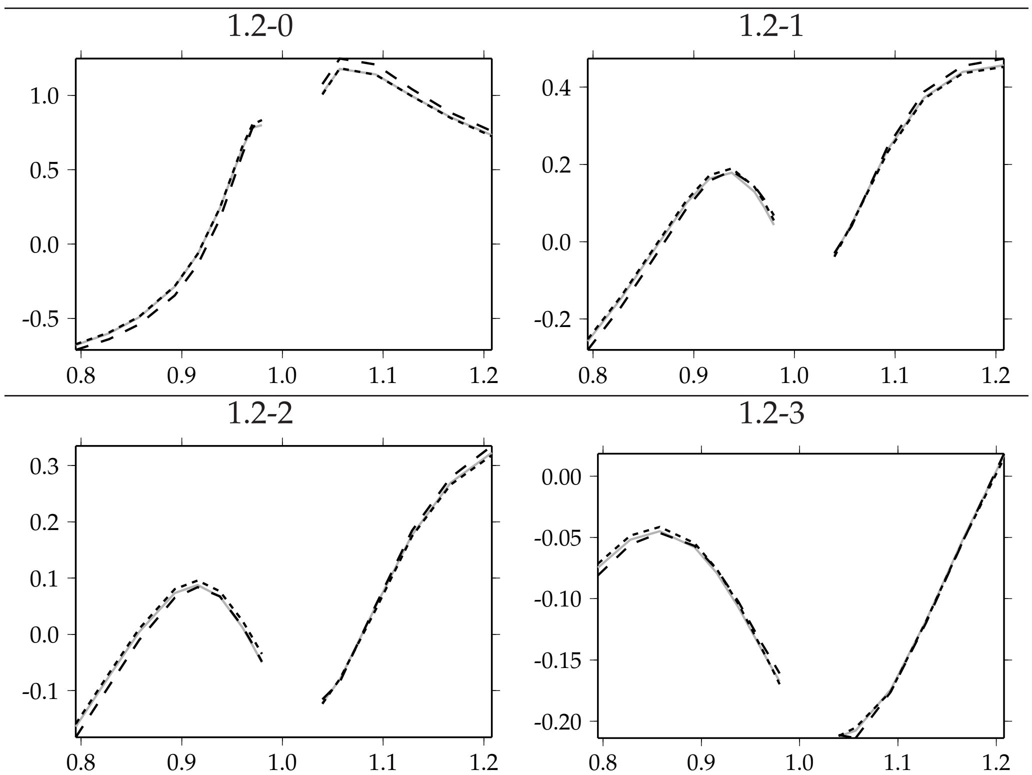


Figure 5.5. Graphs of τ^{11}, view 1, part 3: Detailed group slices.

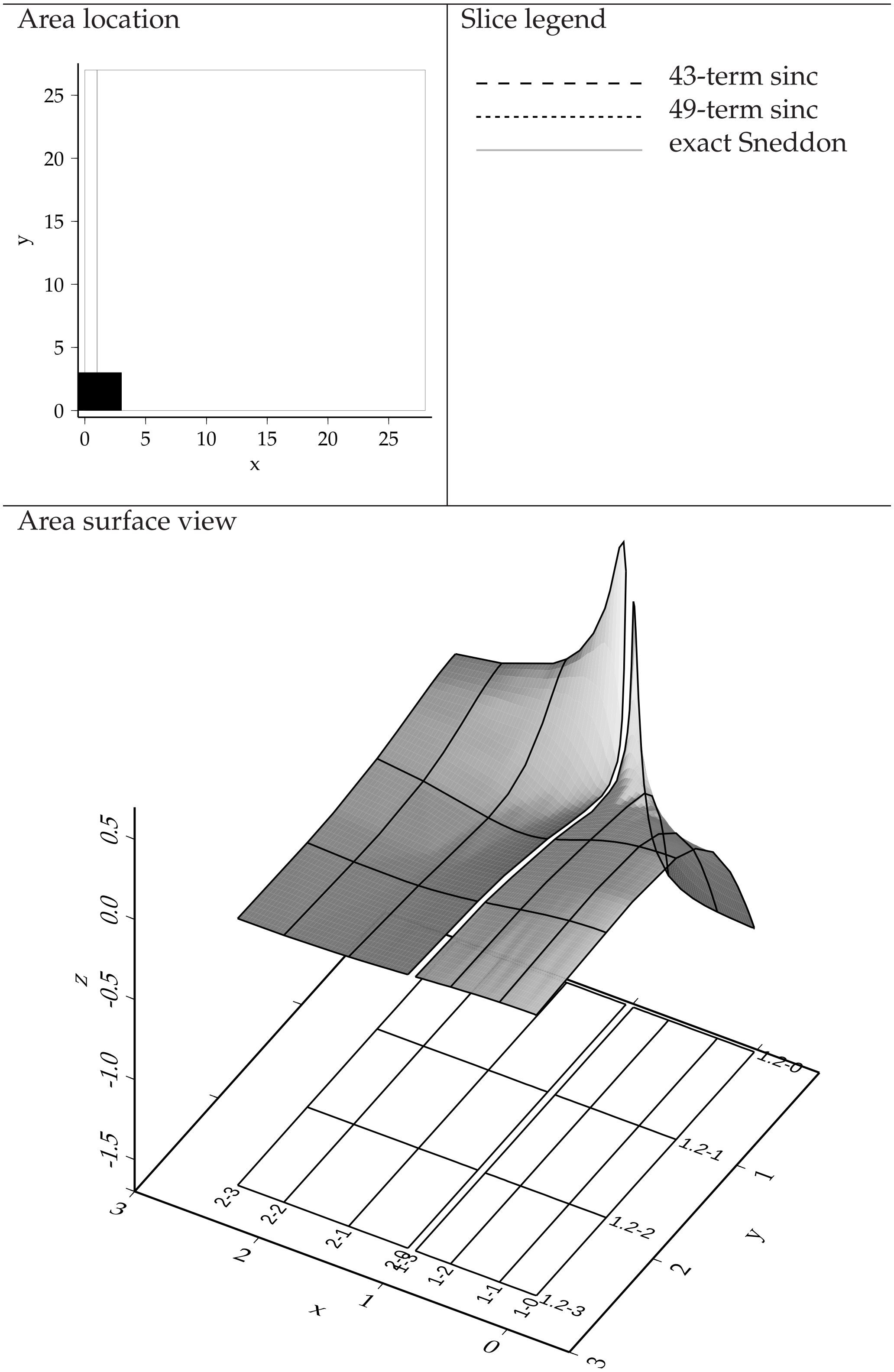


Figure 5.6. Graphs of τ^{11}, view 1, part 1: Location, legends and surface.

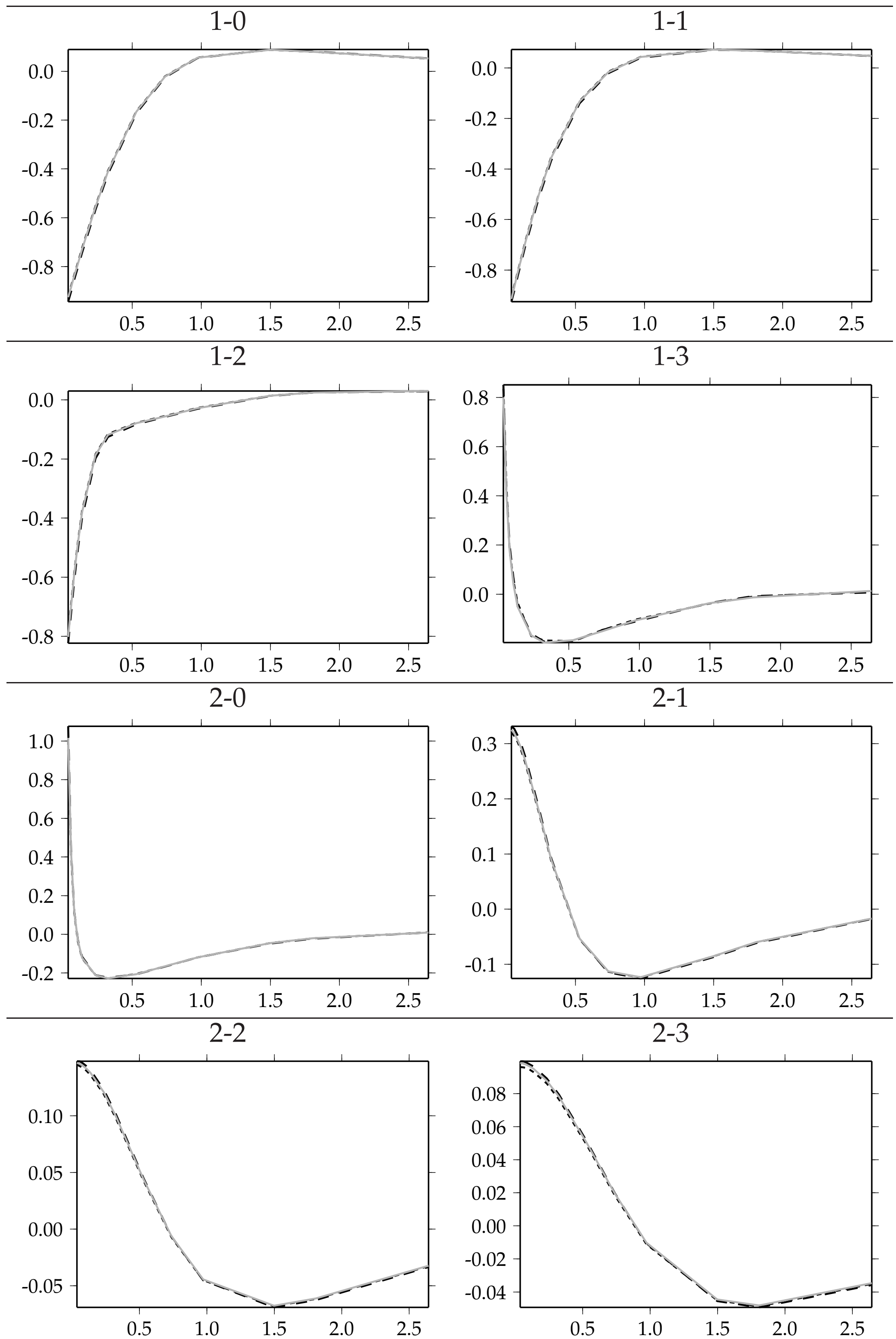


Figure 5.7. Graphs of τ^{11}, view 1, part 2: Detailed single slices.

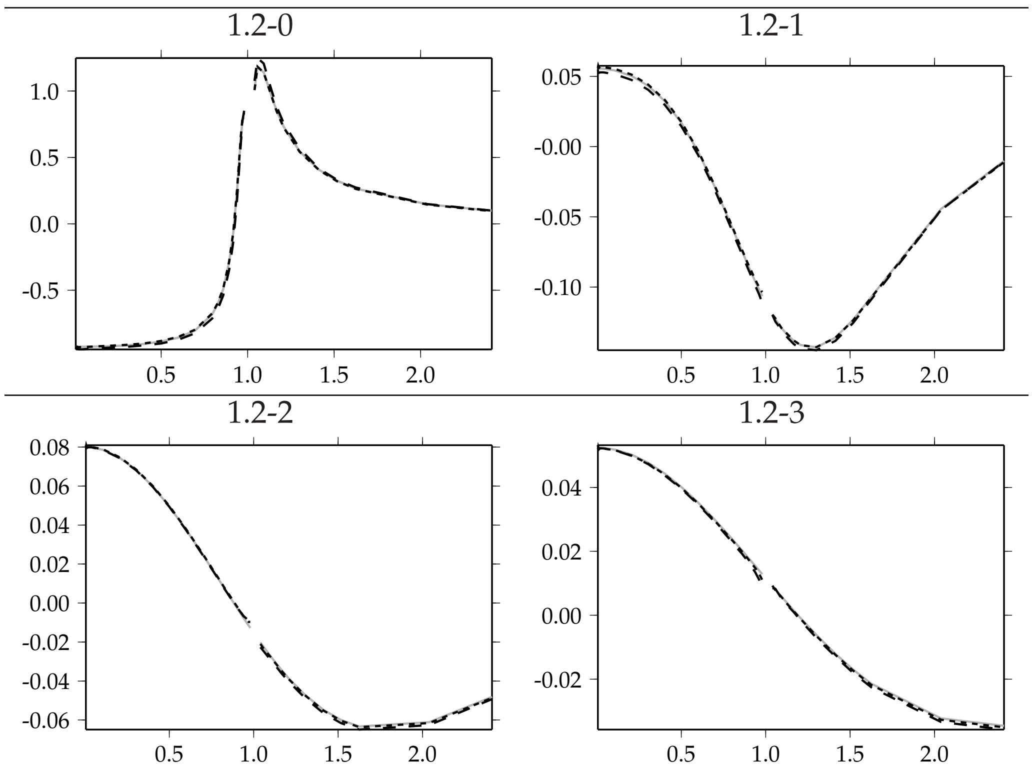


Figure 5.8. Graphs of τ^{11}, view 1, part 3: Detailed group slices.

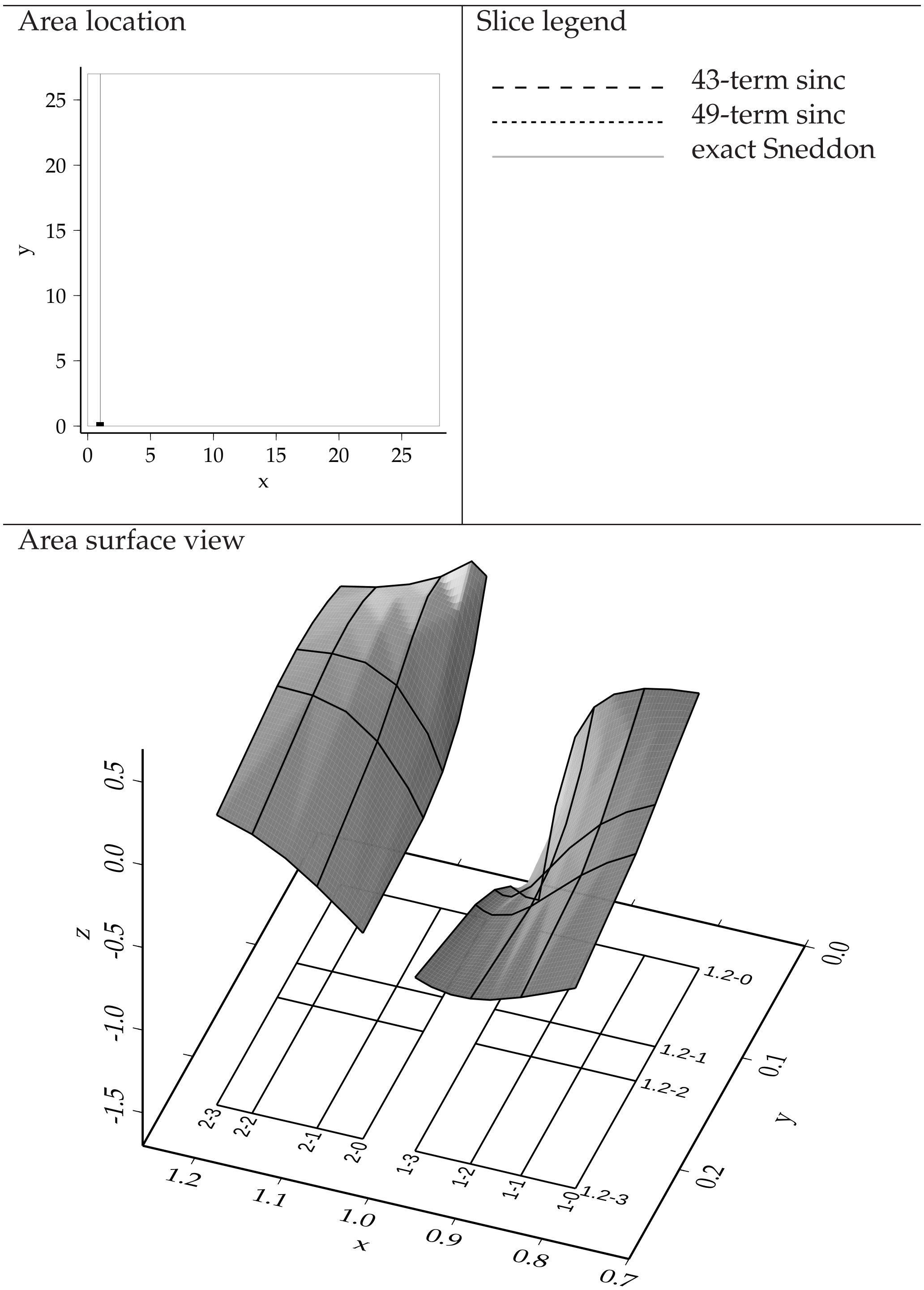


Figure 5.9. Graphs of τ^{12}, view 1, part 1: Location, legends and surface.

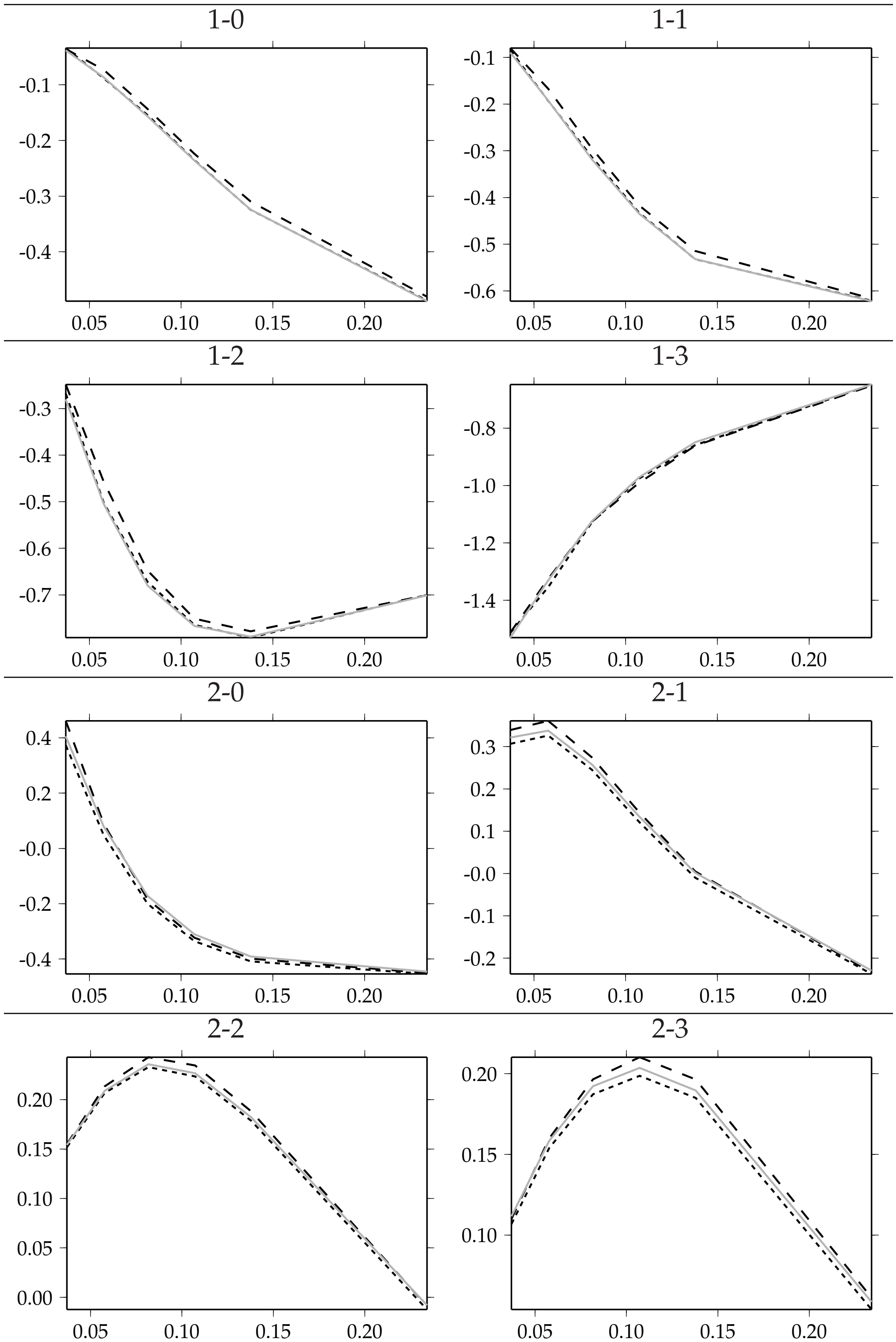


Figure 5.10. Graphs of τ^{12}, view 1, part 2: Detailed single slices.

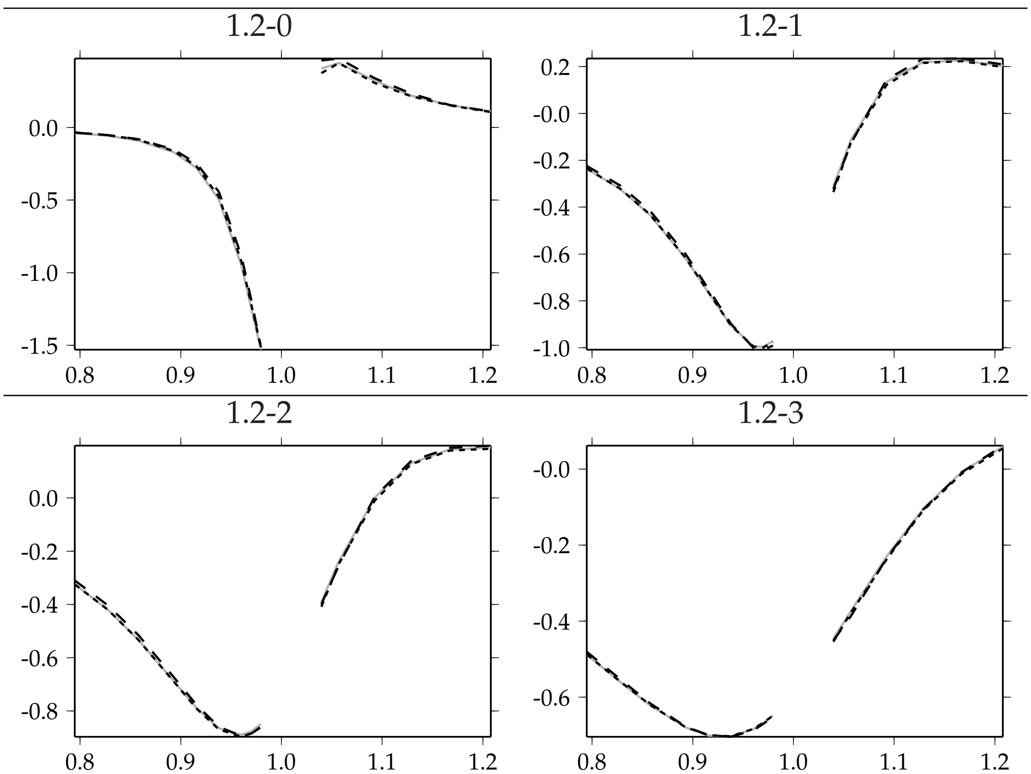


Figure 5.11. Graphs of τ^{12}, view 1, part 3: Detailed group slices.

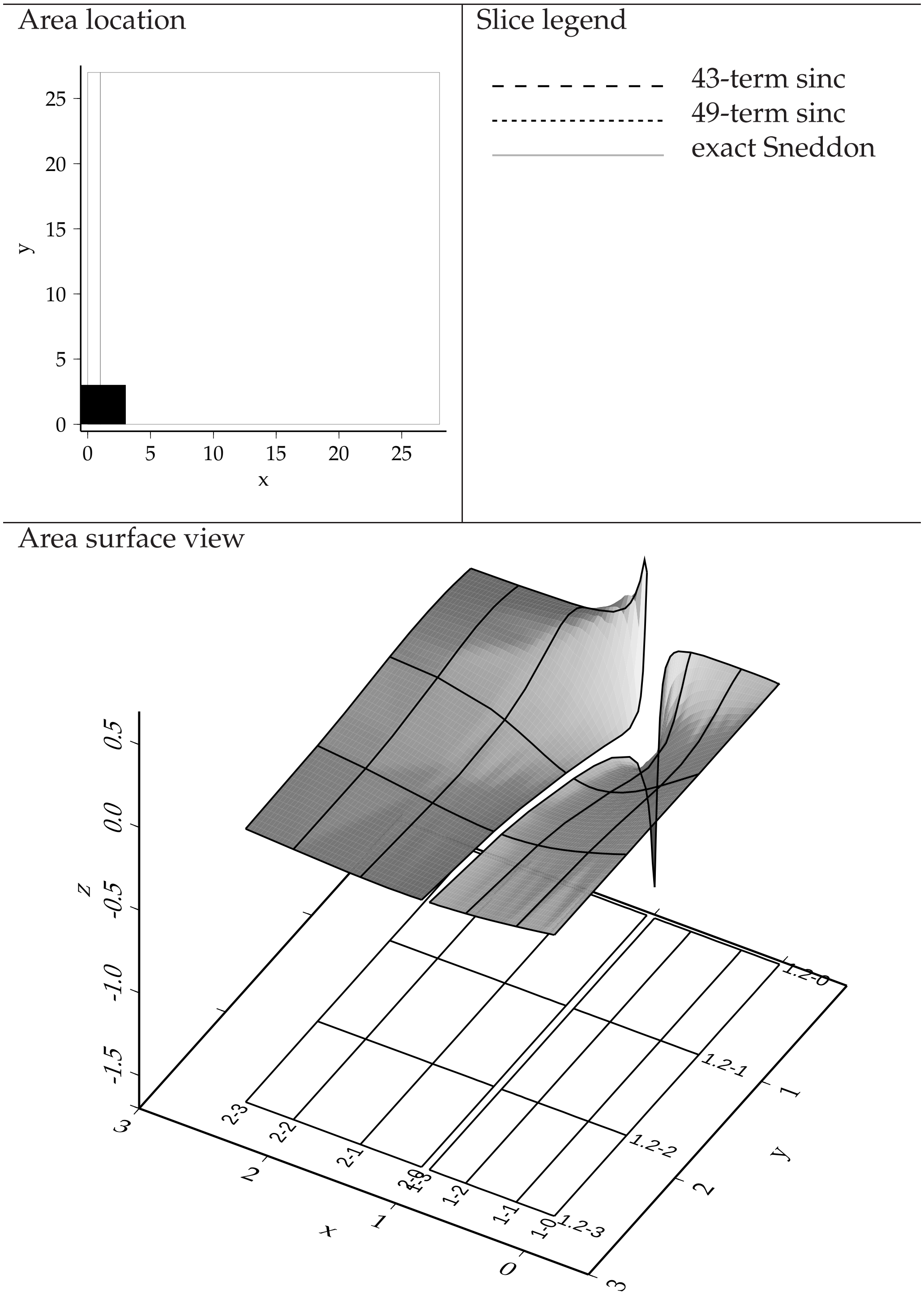


Figure 5.12. Graphs of τ^{12}, view 1, part 1: Location, legends and surface.

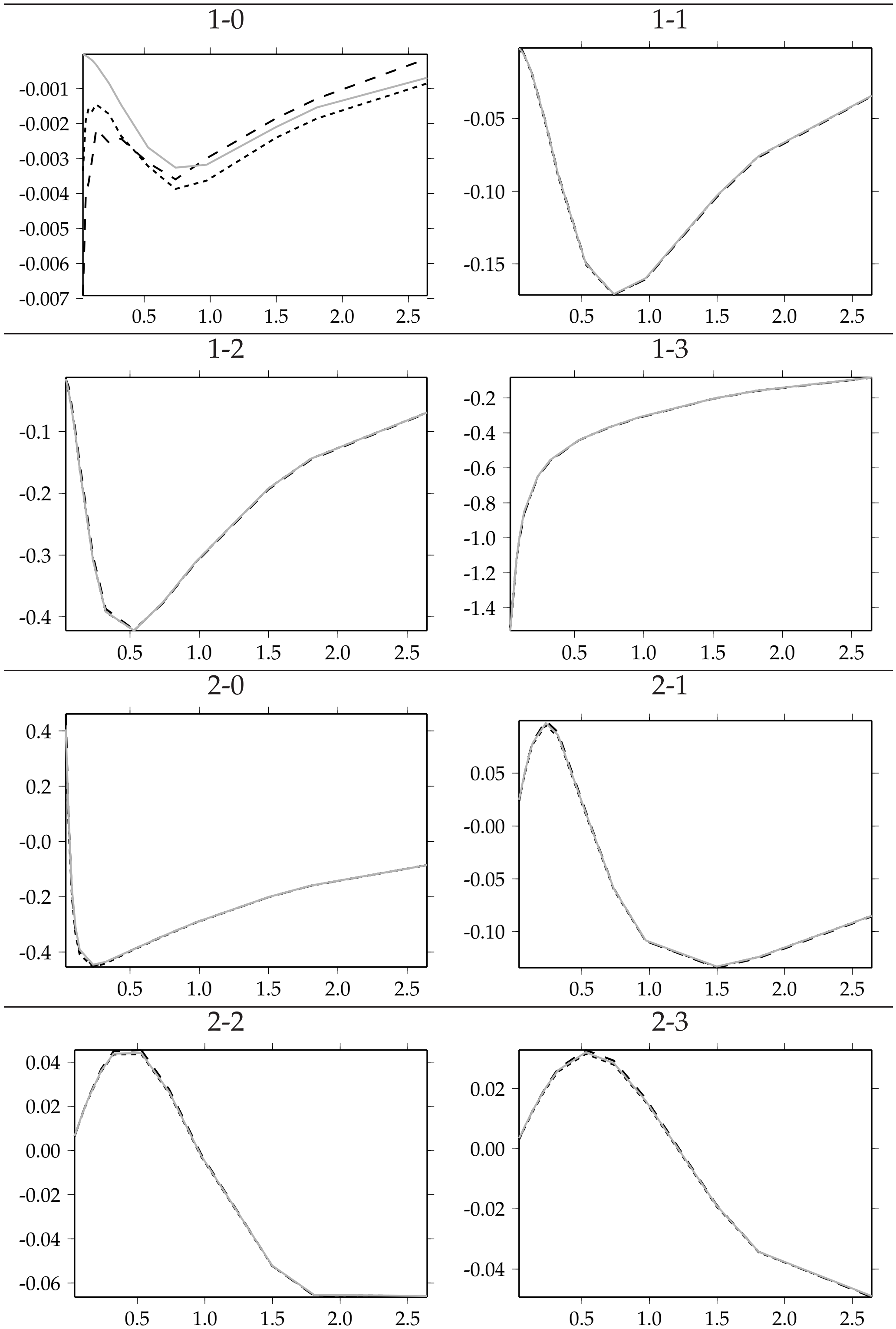


Figure 5.13. Graphs of τ^{12}, view 1, part 2: Detailed single slices.

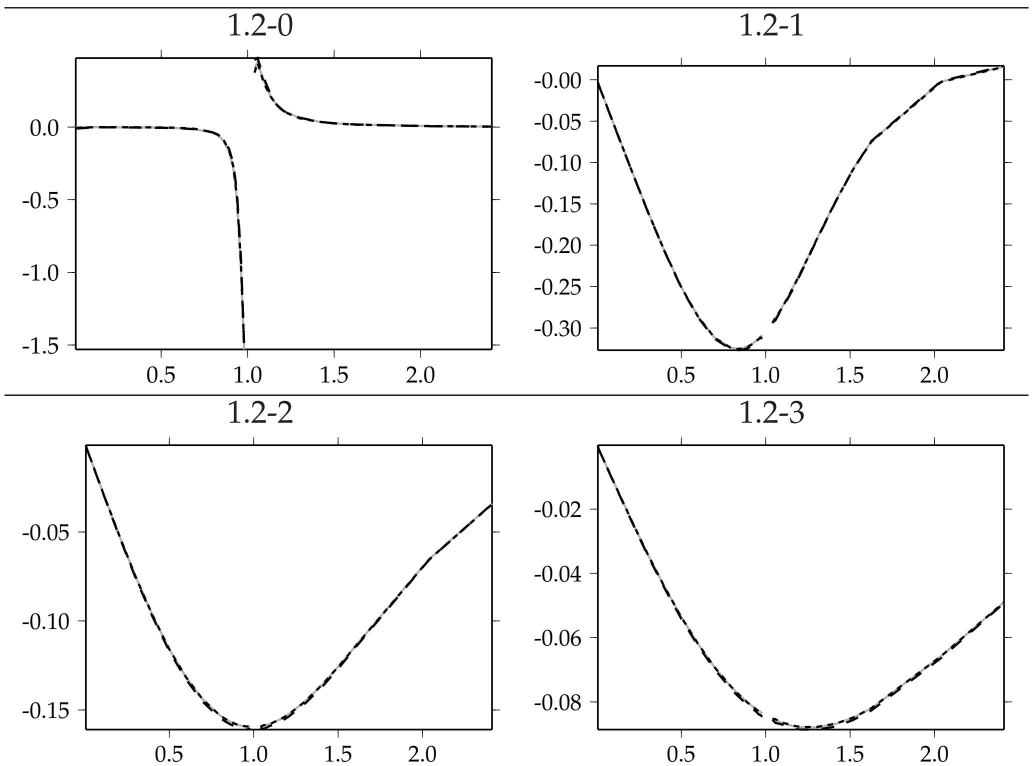


Figure 5.14. Graphs of τ^{12}, view 1, part 3: Detailed group slices.

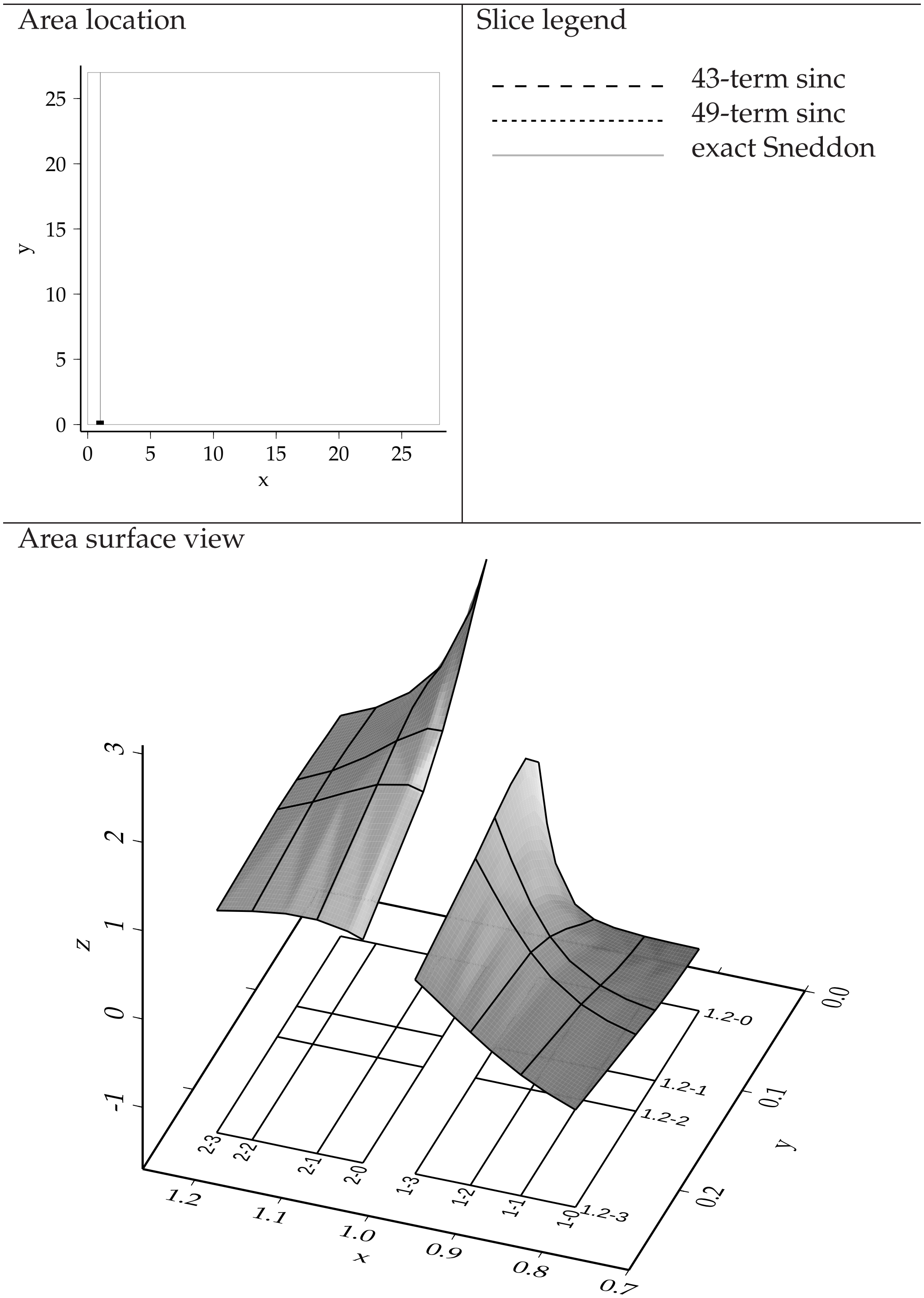


Figure 5.15. Graphs of τ^{22}, view 1, part 1: Location, legends and surface.

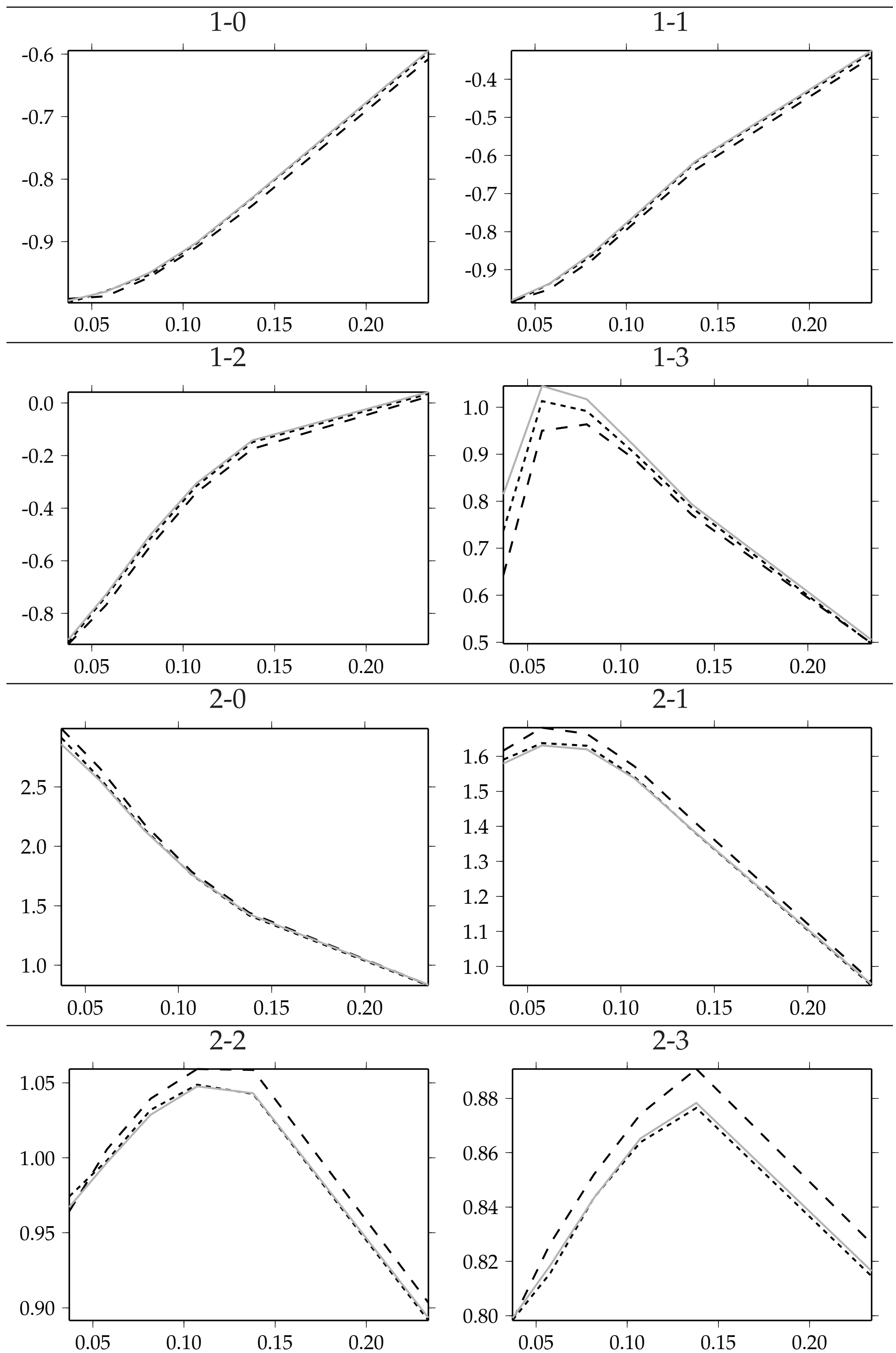


Figure 5.16. Graphs of τ^{22}, view 1, part 2: Detailed single slices.

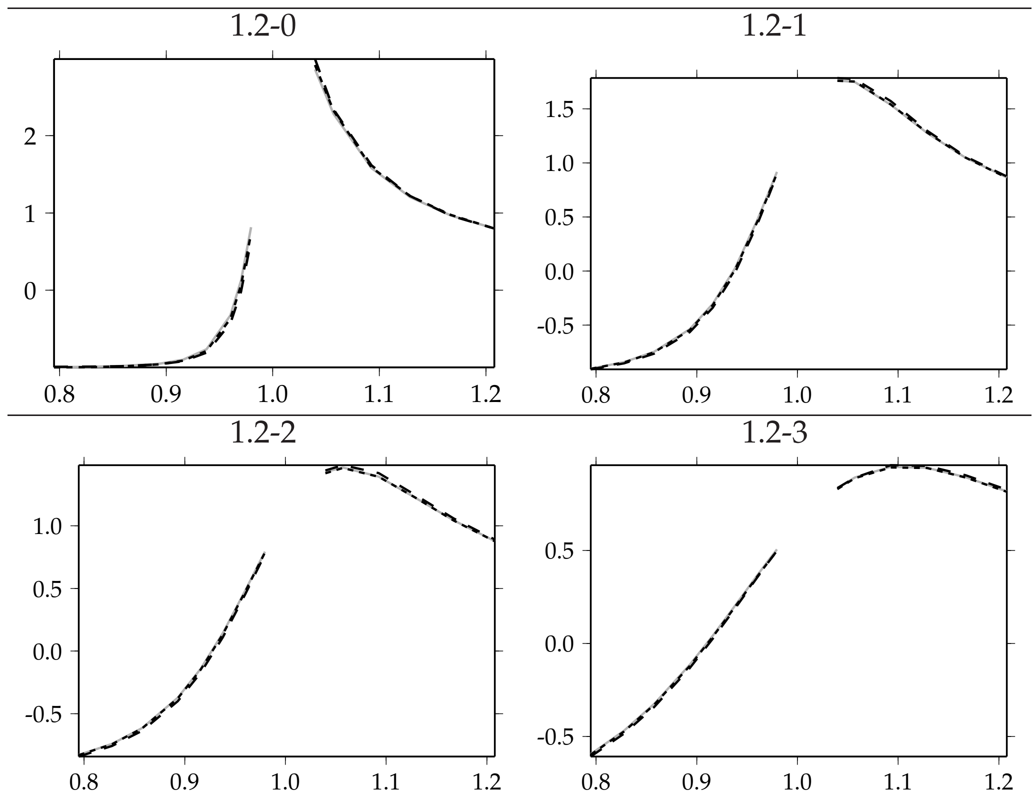


Figure 5.17. Graphs of τ^{22}, view 1, part 3: Detailed group slices.

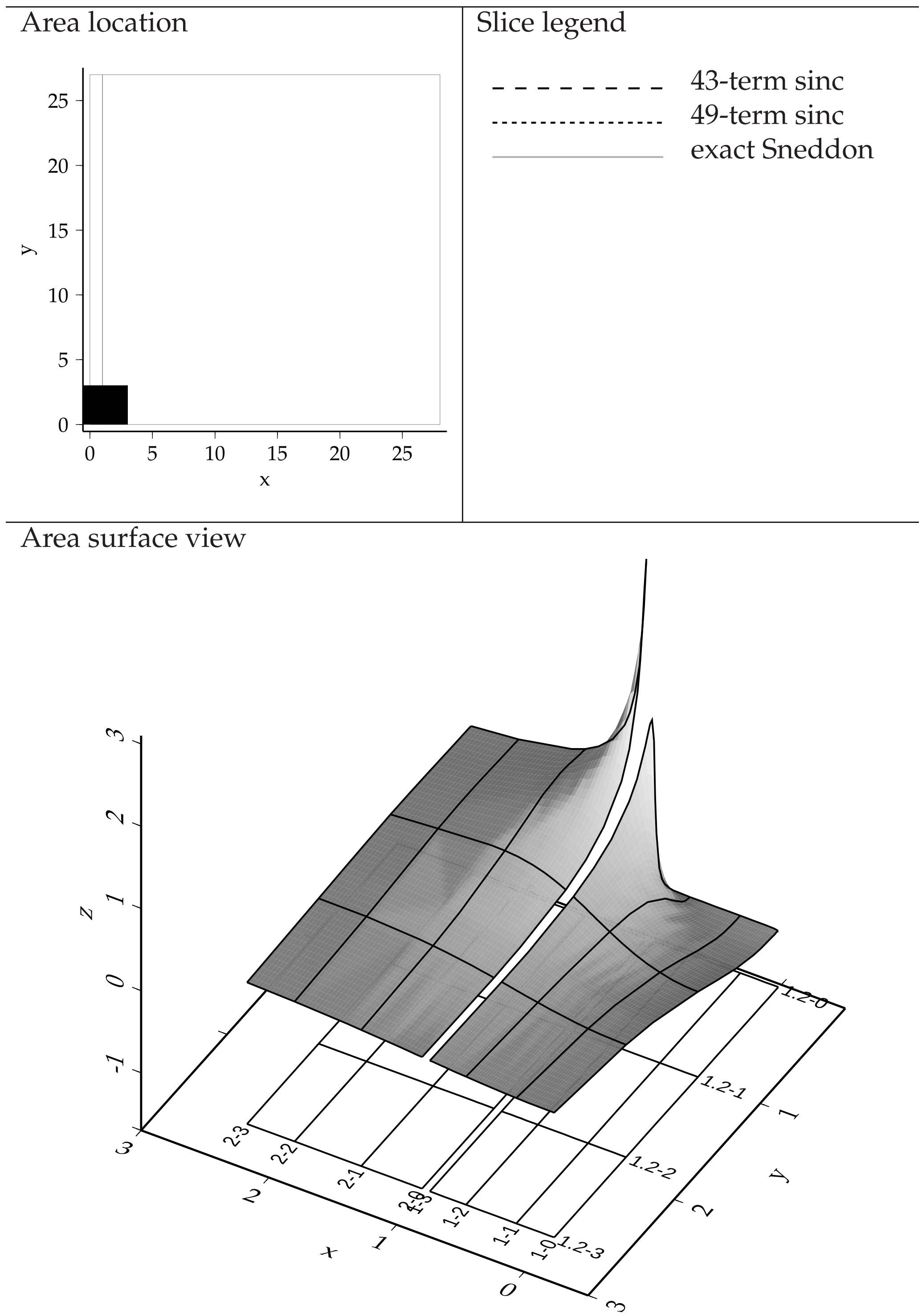


Figure 5.18. Graphs of τ^{22}, view 1, part 1: Location, legends and surface.

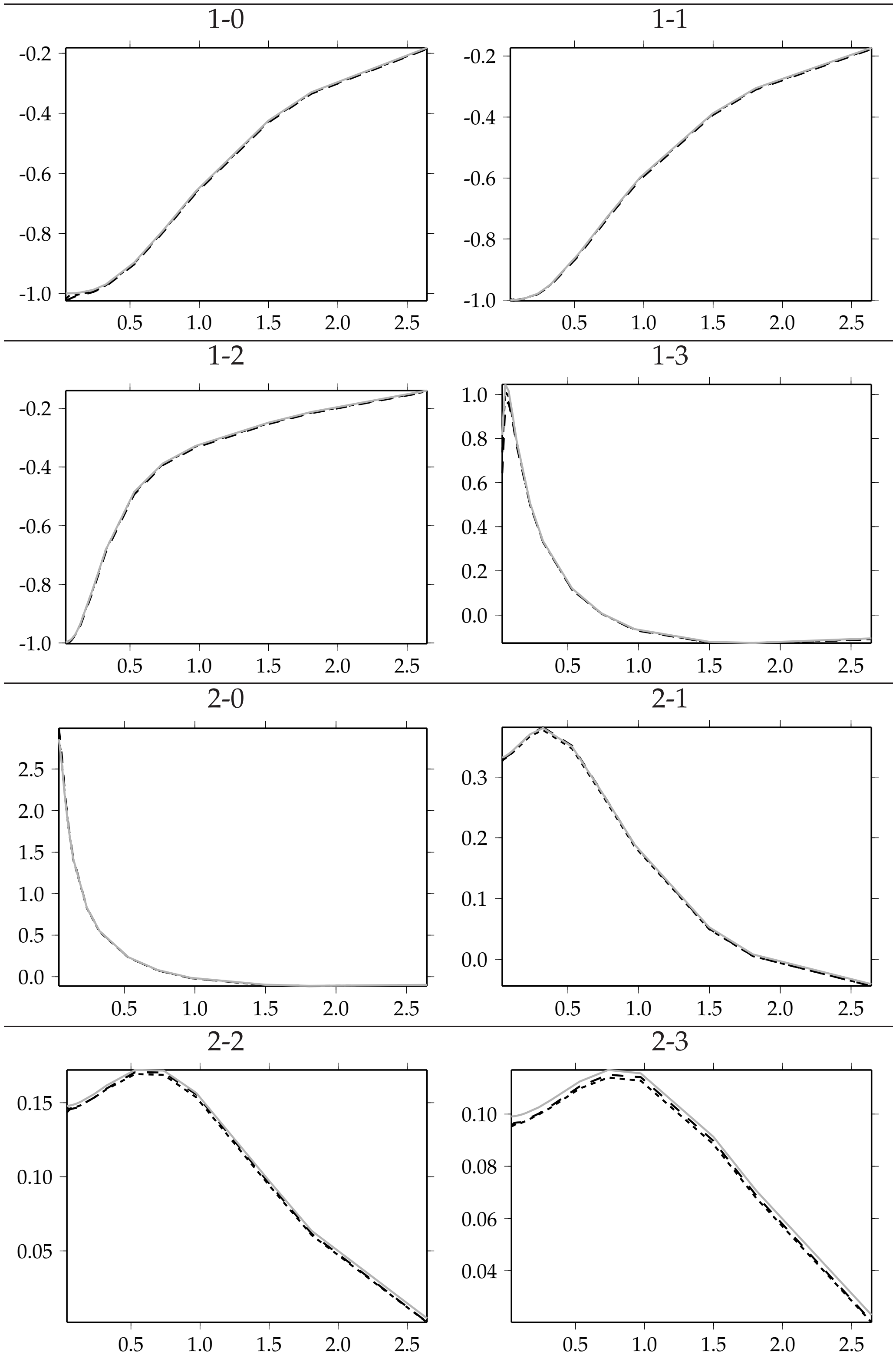


Figure 5.19. Graphs of τ^{22}, view 1, part 2: Detailed single slices.

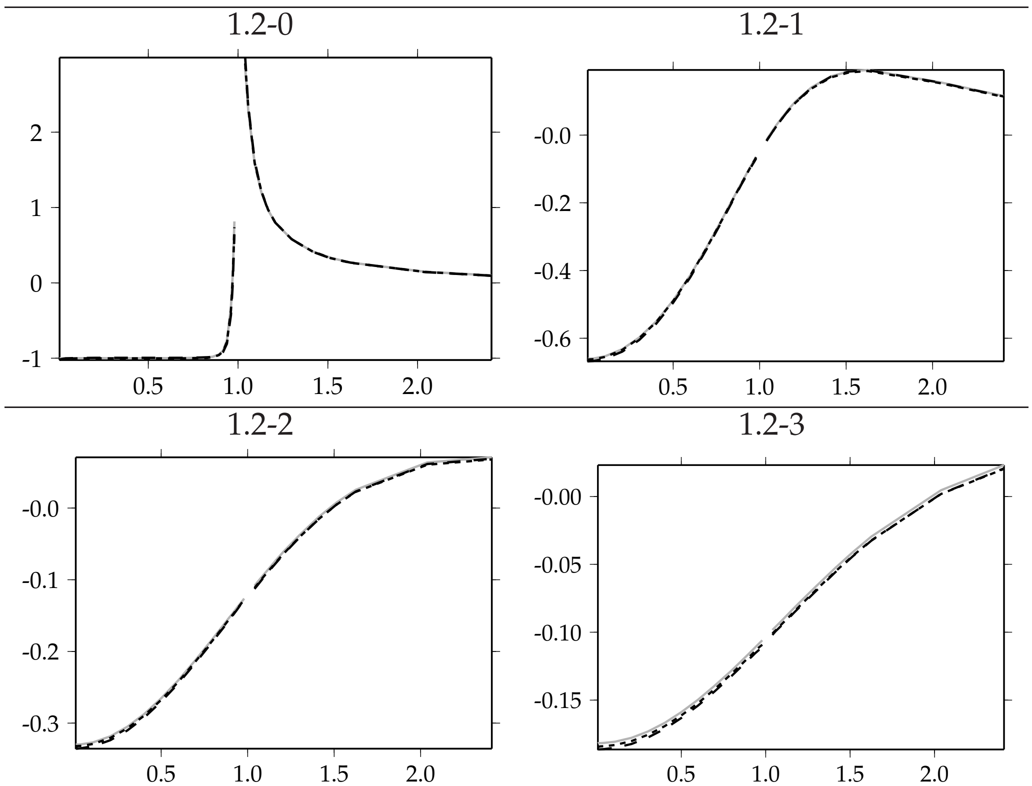


Figure 5.20. Graphs of τ^{22}, view 1, part 3: Detailed group slices.

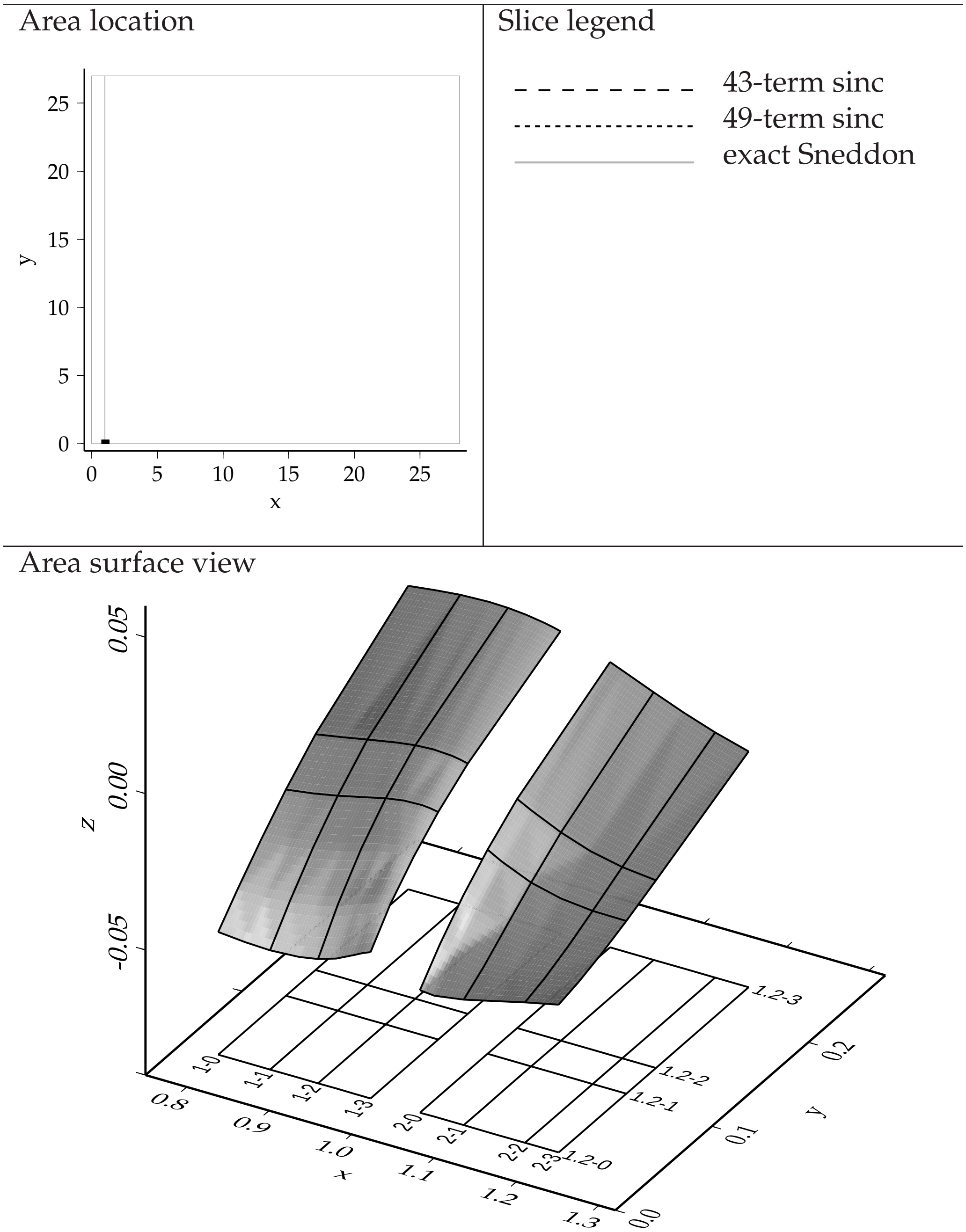


Figure 5.21. Graphs of u[1], view 1, part 1: Location, legends and surface.

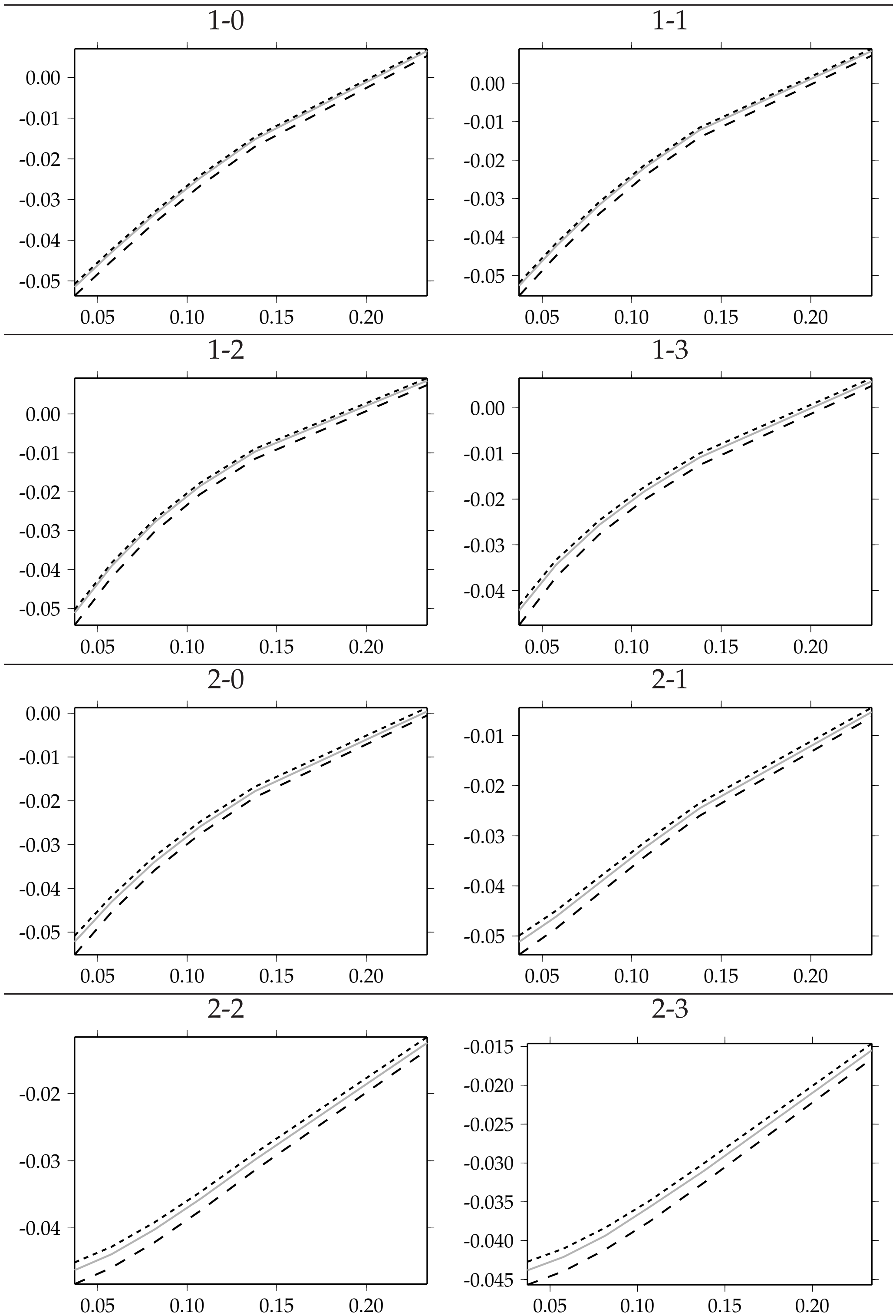


Figure 5.22. Graphs of u^1, view 1, part 2: Detailed single slices.

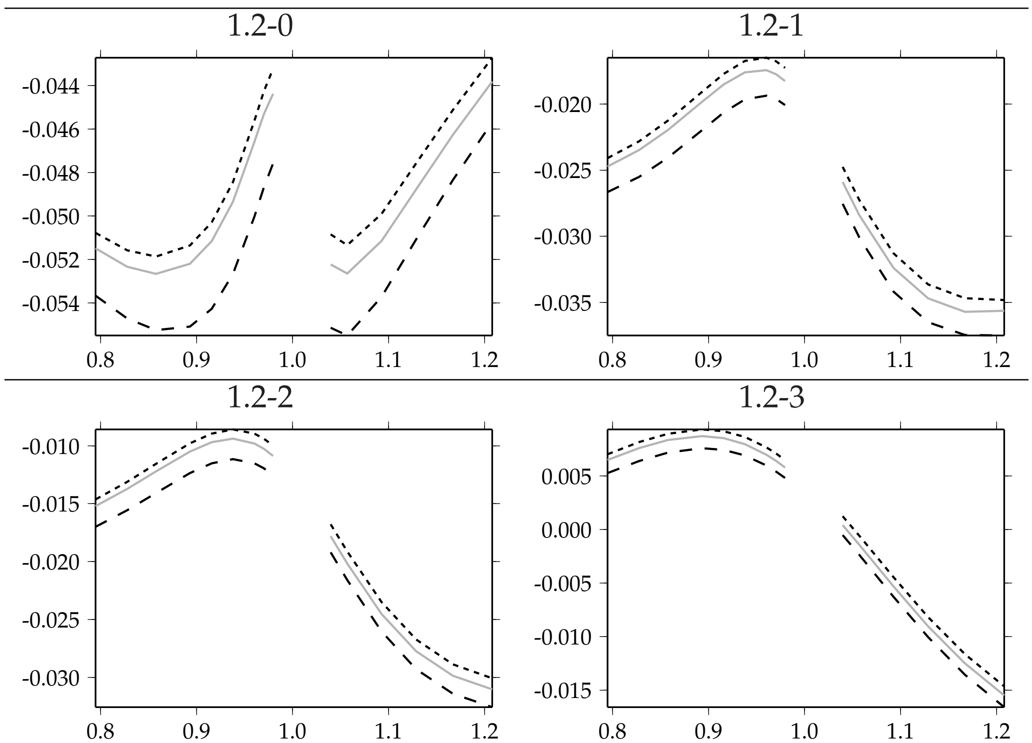


Figure 5.23. Graphs of u^1, view 1, part 3: Detailed group slices.

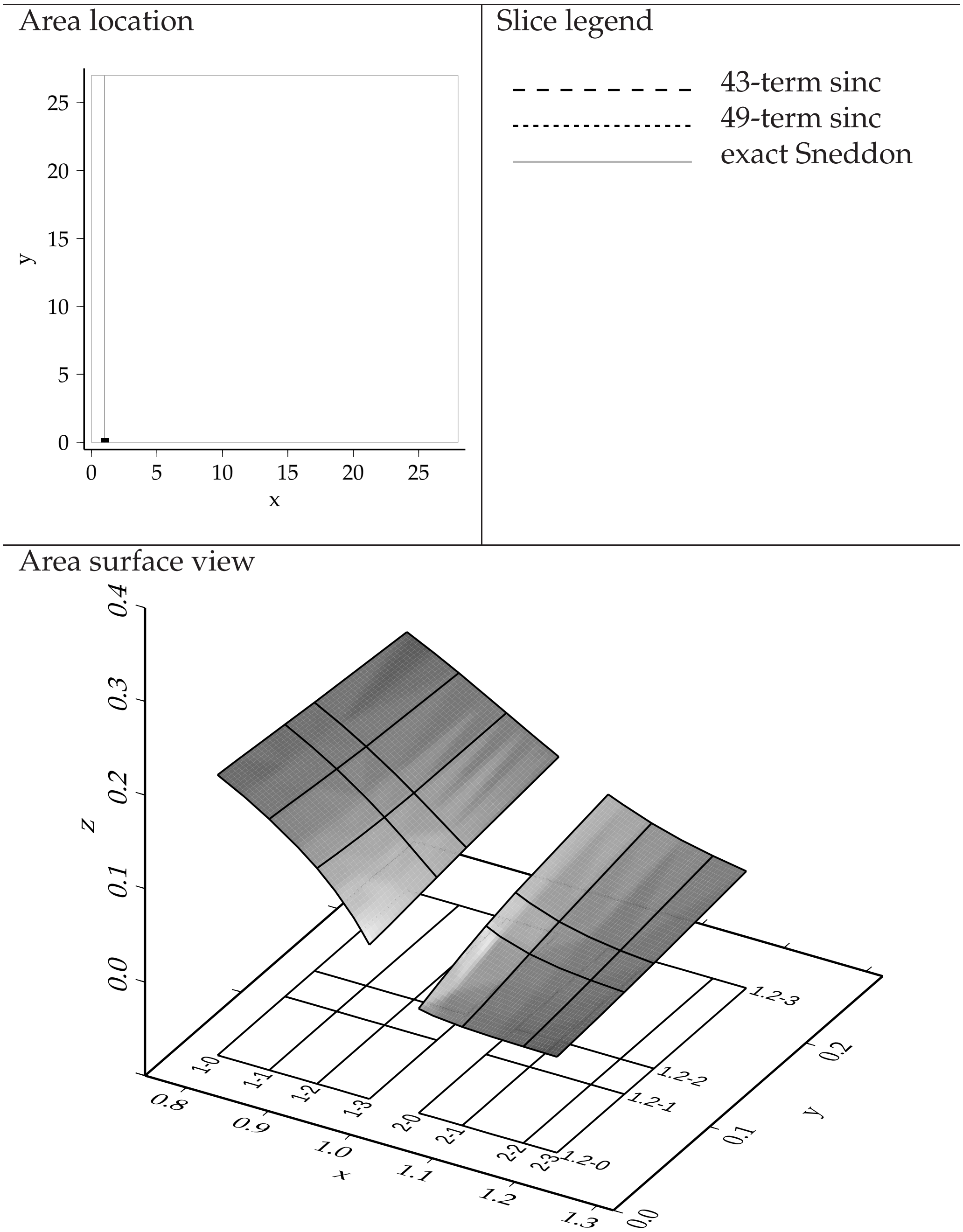


Figure 5.24. Graphs of u^2, view 1, part 1: Location, legends and surface.

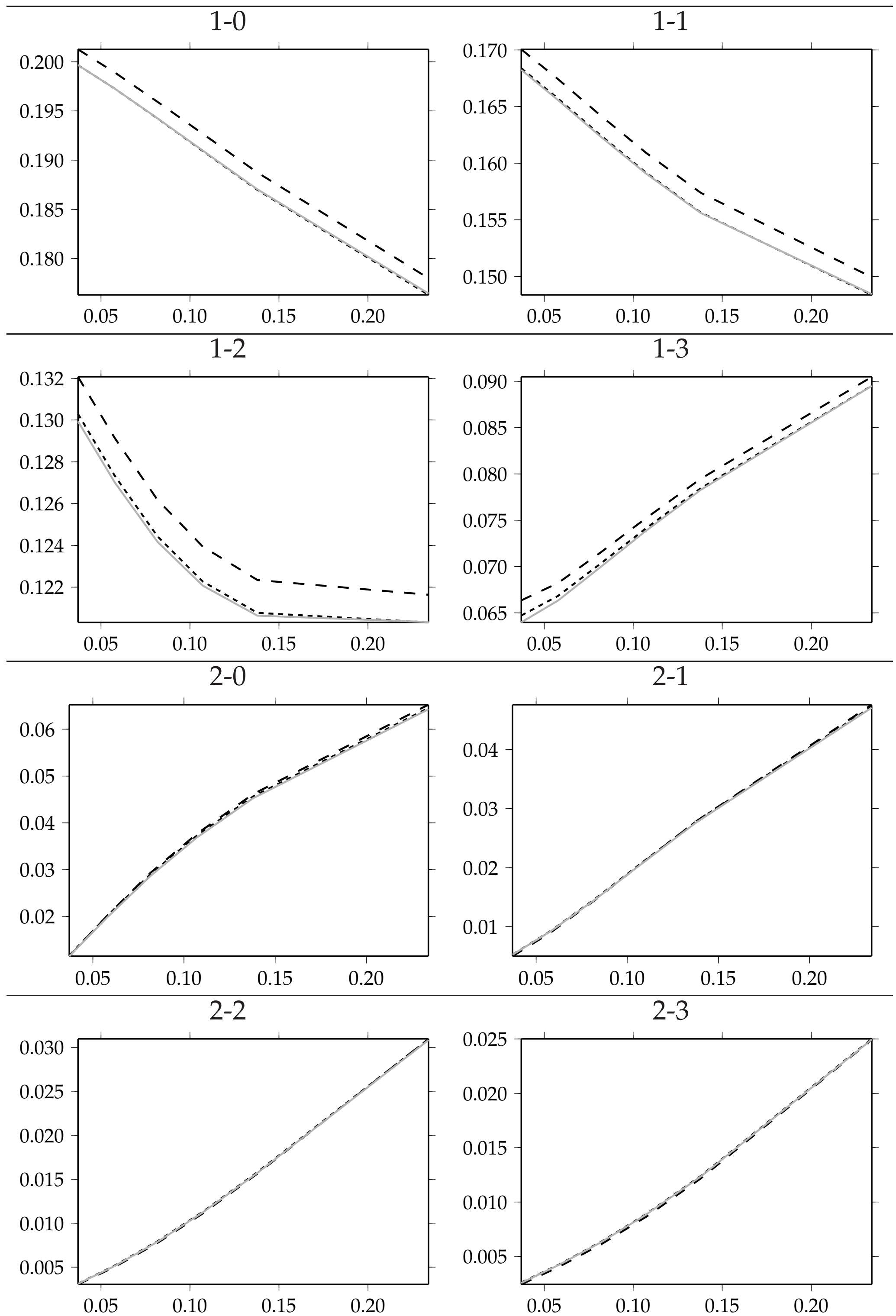


Figure 5.25. Graphs of u^2, view 1, part 2: Detailed single slices.

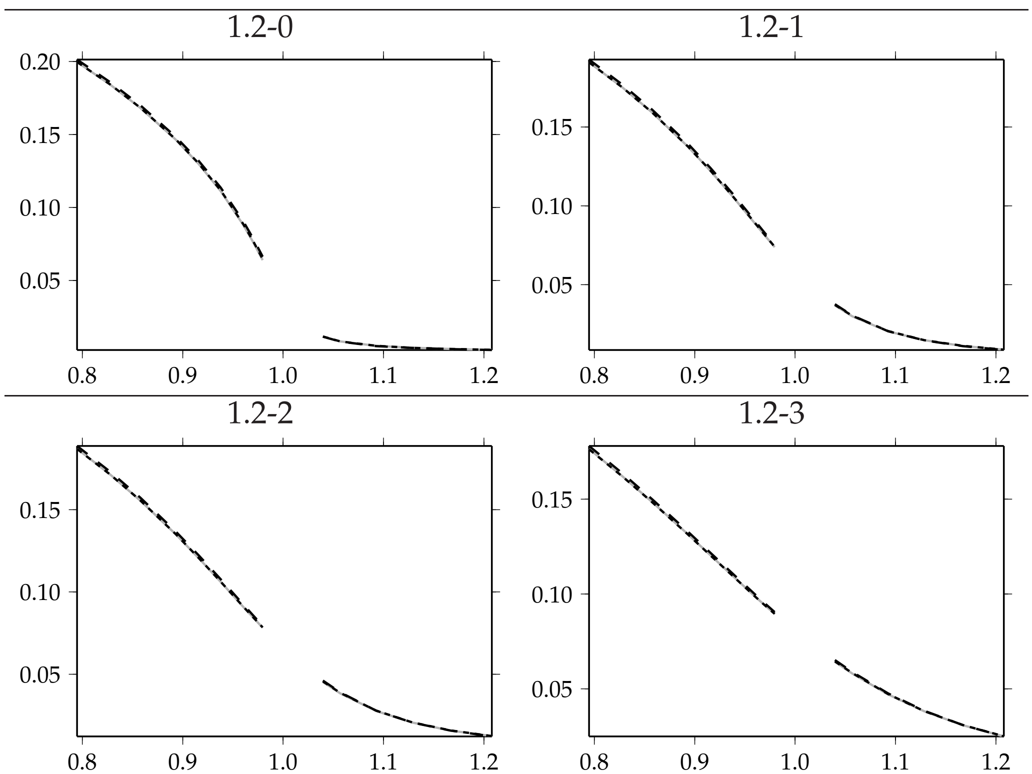


Figure 5.26. Graphs of u^2, view 1, part 3: Detailed group slices.

determined experimentally, and here is found to be 0.03 units from the boundary. Thus, Figure 5.5 and other group slice graphs have a total center gap of length 0.06. This gap should cause no practical problems when viewing stresses, and when using these stresses in further computation, e.g., sinc-based integrals, the singularities (and hence gaps) are suppressed through the presence of a $1/\phi'$ term in the integration formula.

Aside from these restrictions, the pointwise convergence is excellent and shows piecewise smooth solutions, as expected.

5.2 Bimaterial crack

These are the numerical results obtained for the problem of Section 3.2, using the exact answer given by (Rice and Sih 1965) for comparisons in appropriate regions (i.e., those near the crack). The full equations solved here are those shown graphically in Figure 3.4, which in turn use the expansions in Equations 3.13 and 3.14 to result in the following equations, written in the PDE system format of Section 4.5.1, with common system equations factored for clarity.

$$
\begin{array}{l}
\mathsf{equation\_specs}= [\\
\quad \mathsf{unknowns}= [\ \mathsf{u}_1, \quad \mathsf{u}_2, \quad \mathsf{u}_{11}, \quad \mathsf{u}_{12}, \quad \mathsf{u}_{21}, \quad \mathsf{u}_{22}],\\
\quad \mathsf{domains}= [\\
\qquad 1 = [\\
\qquad\quad \mathsf{regions}= [\\
\qquad\qquad \mathsf{Interior}= [\\
\qquad\qquad\quad (((-2)\nu+2)\odot \mathsf{u}_{11}\backslash_1 \oplus (1-2\nu)\odot \mathsf{u}_{12}\backslash_2)\oplus\\
\qquad\qquad\qquad 1\odot \mathsf{u}_{21}\backslash_2 \cong 0\ ,\\
\qquad\qquad\quad ((1-2\nu)\odot \mathsf{u}_{21}\backslash_1 \oplus ((-2)\nu+2)\odot \mathsf{u}_{22}\backslash_2)\oplus\\
\qquad\qquad\qquad 1\odot \mathsf{u}_{11}\backslash_2 \cong 0\ ,\\
\qquad\qquad\quad \mathsf{system}],\\
\qquad\qquad \mathsf{Bottom}= [\\
\qquad\qquad\quad -\mu\odot \mathsf{u}_1\backslash_2 \oplus -\mu\odot \mathsf{u}_2\backslash_1 \cong 0\ ,\\
\qquad\qquad\quad \dfrac{2\mu(1-\nu)}{(-1)+2\nu}\odot \mathsf{u}_2\backslash_2\oplus\\
\qquad\qquad\qquad \dfrac{2\mu\nu}{(-1)+2\nu}\odot \mathsf{u}_1\backslash_1 \cong \sigma\ ,\\
\qquad\qquad\quad \mathsf{system}],\\
\qquad\qquad \mathsf{Left}= [\ 1\odot \mathsf{u}_{21} \cong 0\ , \quad 1\odot \mathsf{u}_1 \cong 0\ , \quad \mathsf{system}],\\
\qquad\qquad \mathsf{Top}= [\ 1\odot \mathsf{u}_1 \cong 0\ , \quad 1\odot \mathsf{u}_2 \cong 0\ , \quad \mathsf{system}],
\end{array}
$$

Right= [$1 \odot$ $u_{11} \cong 0$, $1 \odot u_{21} \cong 0$, system],
RightOL= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$]]],
2 = [
regions= [
LeftOL= [$(-1) \odot u_{11} \cong 0$, $(-1) \odot u_{21} \cong 0$],
Left= [$(-1) \odot u_1 \cong 0$, $(-1) \odot u_2 \cong 0$,
system],
Top= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$, system],
Right= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$, system],
Interior= [
$(((-2)\nu + 2) \odot u_{11}\backslash_1 \oplus (1-2\nu) \odot u_{12}\backslash_2) \oplus$
$1 \odot u_{21}\backslash_2 \cong 0$,
$((1-2\nu) \odot u_{21}\backslash_1 \oplus ((-2)\nu + 2) \odot u_{22}\backslash_2) \oplus$
$1 \odot u_{11}\backslash_2 \cong 0$,
system],
Bottom= [
$-\mu \odot u_1\backslash_2 \oplus -\mu \odot u_2\backslash_1 \cong 0$,
$\dfrac{2\mu(1-\nu)}{(-1)+2\nu} \odot u_2\backslash_2 \oplus$
$\dfrac{2\mu\nu}{(-1)+2\nu} \odot u_1\backslash_1 \cong 0$,
system],
BottomOL= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$]]],
3 = [
regions= [
TopOL= [
$\mu \odot u_1\backslash_2 \oplus \mu \odot u_2\backslash_1 \cong 0$,
$\dfrac{(-2)\mu(1-\nu)}{(-1)+2\nu} \odot u_2\backslash_2 \oplus$
$\dfrac{(-2)\mu\nu}{(-1)+2\nu} \odot u_1\backslash_1 \cong 0$],
Top= [$(-1) \odot u_1 \cong 0$, $(-1) \odot u_2 \cong 0$,
system],
Right= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$, system],
Bottom= [$1 \odot u_1 \cong 0$, $1 \odot u_2 \cong 0$, system],
Interior= [
$(((-2)\nu + 2) \odot u_{11}\backslash_1 \oplus (1-2\nu) \odot u_{12}\backslash_2) \oplus$
$1 \odot u_{21}\backslash_2 \cong 0$,
$((1-2\nu) \odot u_{21}\backslash_1 \oplus ((-2)\nu + 2) \odot u_{22}\backslash_2) \oplus$
$1 \odot u_{11}\backslash_2 \cong 0$,
system],
Left= [$(-1) \odot u_1 \cong 0$, $(-1) \odot u_2 \cong 0$,
system],
LeftOL= [$(-1) \odot u_{11} \cong 0$, $(-1) \odot u_{21} \cong 0$]]],
4 = [

$$
\begin{array}{l}
\mathsf{regions} = [\\
\quad \mathsf{RightOL} = [\, 1 \odot \mathsf{u}_1 \cong 0\,, \quad 1 \odot \mathsf{u}_2 \cong 0\,],\\
\quad \mathsf{Right} = [\, 1 \odot \mathsf{u}_{11} \cong 0\,, \quad 1 \odot \mathsf{u}_{21} \cong 0\,, \quad \mathsf{system}],\\
\quad \mathsf{Bottom} = [\, 1 \odot \mathsf{u}_1 \cong 0\,, \quad 1 \odot \mathsf{u}_2 \cong 0\,, \quad \mathsf{system}],\\
\quad \mathsf{Left} = [\, 1 \odot \mathsf{u}_{21} \cong 0\,, \quad 1 \odot \mathsf{u}_1 \cong 0\,, \quad \mathsf{system}],\\
\quad \mathsf{Top} = [\\
\qquad \mu \odot \mathsf{u}_1\backslash_2 \oplus \mu \odot \mathsf{u}_2\backslash_1 \cong 0\,,\\
\qquad \dfrac{(-2)\mu(1-\nu)}{(-1)+2\nu} \odot \mathsf{u}_2\backslash_2 \oplus\\
\qquad\quad \dfrac{(-2)\mu\nu}{(-1)+2\nu} \odot \mathsf{u}_1\backslash_1 \cong -\sigma\,,\\
\qquad \mathsf{system}],\\
\quad \mathsf{Interior} = [\\
\qquad (((-2)\nu+2) \odot \mathsf{u}_{11}\backslash_1 \oplus (1-2\nu) \odot \mathsf{u}_{12}\backslash_2) \oplus\\
\qquad\quad 1 \odot \mathsf{u}_{21}\backslash_2 \cong 0\,,\\
\qquad ((1-2\nu) \odot \mathsf{u}_{21}\backslash_1 \oplus ((-2)\nu+2) \odot \mathsf{u}_{22}\backslash_2) \oplus\\
\qquad\quad 1 \odot \mathsf{u}_{11}\backslash_2 \cong 0\,,\\
\qquad \mathsf{system}]]]]]
\end{array}
$$

$$
\begin{array}{l}
\mathsf{system} = [\\
\quad 1 \odot \mathsf{u}_1\backslash_1 \oplus (-1) \odot \mathsf{u}_{11} \cong 0\,,\\
\quad 1 \odot \mathsf{u}_1\backslash_2 \oplus (-1) \odot \mathsf{u}_{12} \cong 0\,,\\
\quad 1 \odot \mathsf{u}_2\backslash_1 \oplus (-1) \odot \mathsf{u}_{21} \cong 0\,,\\
\quad 1 \odot \mathsf{u}_2\backslash_2 \oplus (-1) \odot \mathsf{u}_{22} \cong 0\,]
\end{array}
$$

The numerical parameters are shown in Table 5.2; the general geometry was shown in Figure 3.3 and is shown drawn to scale along with the individual graphs. This bimaterial problem results in much larger memory requirements than the single material problem; because the largest N is only 18, and $N \leq 12$ produces worthless results, a norm-based convergence check is impractical. For this problem, therefore, only a combination of three-dimensional surface and two-dimensional xy-plots are shown. Their structure is the same as in Sec-

Table 5.2. Bi-material problem parameters.

Geometry		Problem Parameters				Sinc Constants	
		Domain 1		Domain 2			
c	1.0	σ	1.0	σ	1.0	α	1.0
w	27.0	ν	0.33	ν	0.33	d	$\pi/2$
h	27.0	μ	1.0	μ	3.21		

tion 5.1.2. As only the stresses are readily computed from the solutions provided in (Rice and Sih 1965), the graphs are further restricted to the stresses τ^{11}, τ^{12} and τ^{22}. Each is shown on two areas in Figures 5.27 through 5.35.

These computed solutions are of no practical use; a larger N is required, but this is not feasible without improving the space requirements of the algorithm. Nevertheless, they illustrate the generality of the method and algorithm, and they do demonstrate convergence in a limited fashion.

The interdomain gap is 0.06, as for the single-material problem.

5.3 Conclusions

The sinc method's convergence rate is $O(\sqrt{N}e^{-g\sqrt{N}})$, so for large N, excellent accuracy is expected. In practice, the accuracy for small N, where the meaning of "small" is problem dependent, is very poor; so poor there may not be much resemblance between the correct solution and the computed one.

This is in sharp contrast to polynomial-based methods, in which even a low-order approximation resembles the correct solution, and accuracy gradually increases (typically linearly) with an increase in the number of terms used in computation.

The approximations of functions and derivatives are uniformly accurate and smooth. This is a tremendous advantage for problems in which both approximations are needed, as other methods usually produce only one nonsmooth solution, and the other must be calculated by inaccurate interpolation.

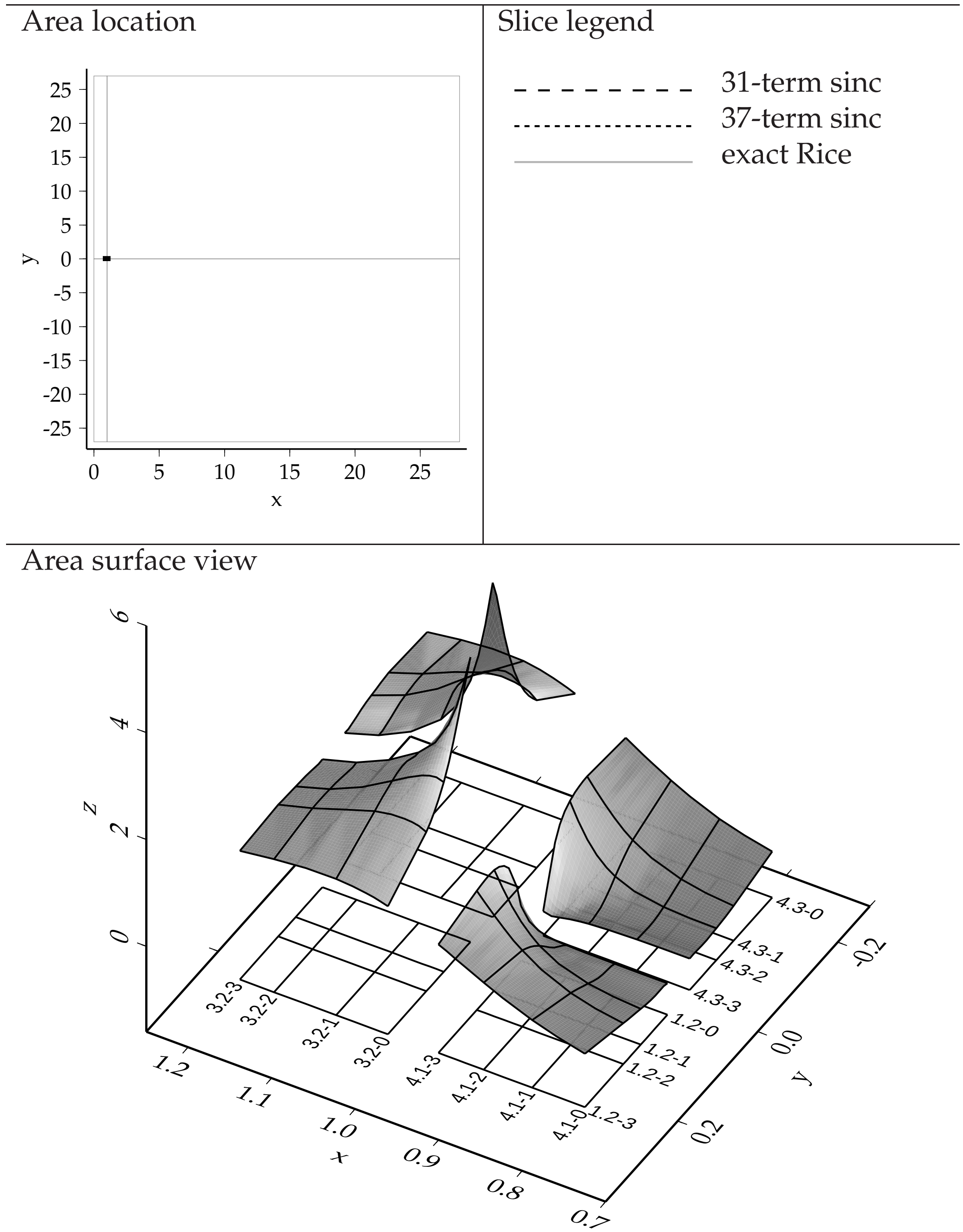


Figure 5.27. Graphs of τ^{22}, view 1, part 1: Location, legends and surface.

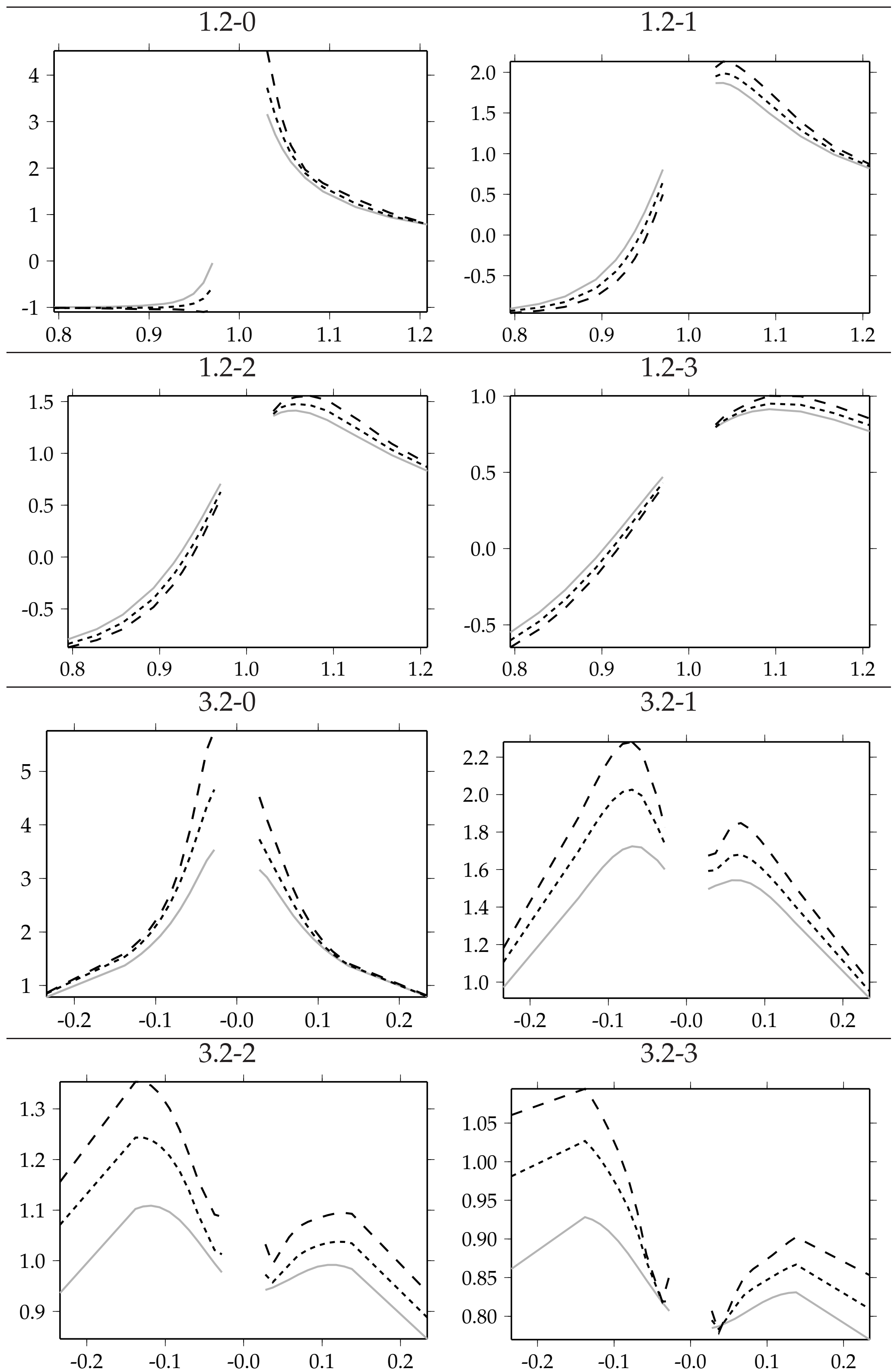


Figure 5.28. Graphs of τ^{22}, view 1, part 2: Detailed group slices.

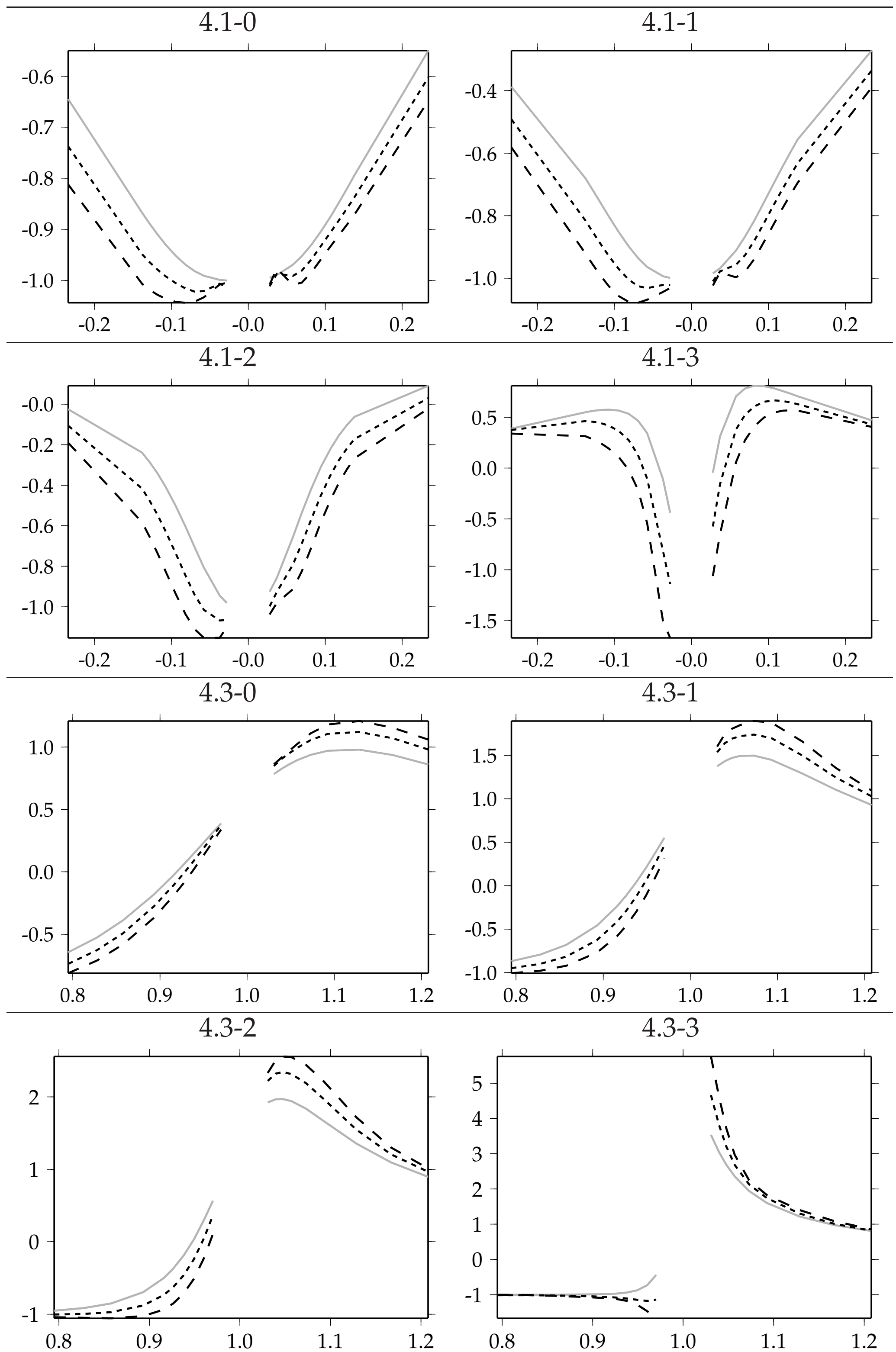


Figure 5.29. Graphs of τ^{22}, view 1, part 3: Detailed group slices, continued.

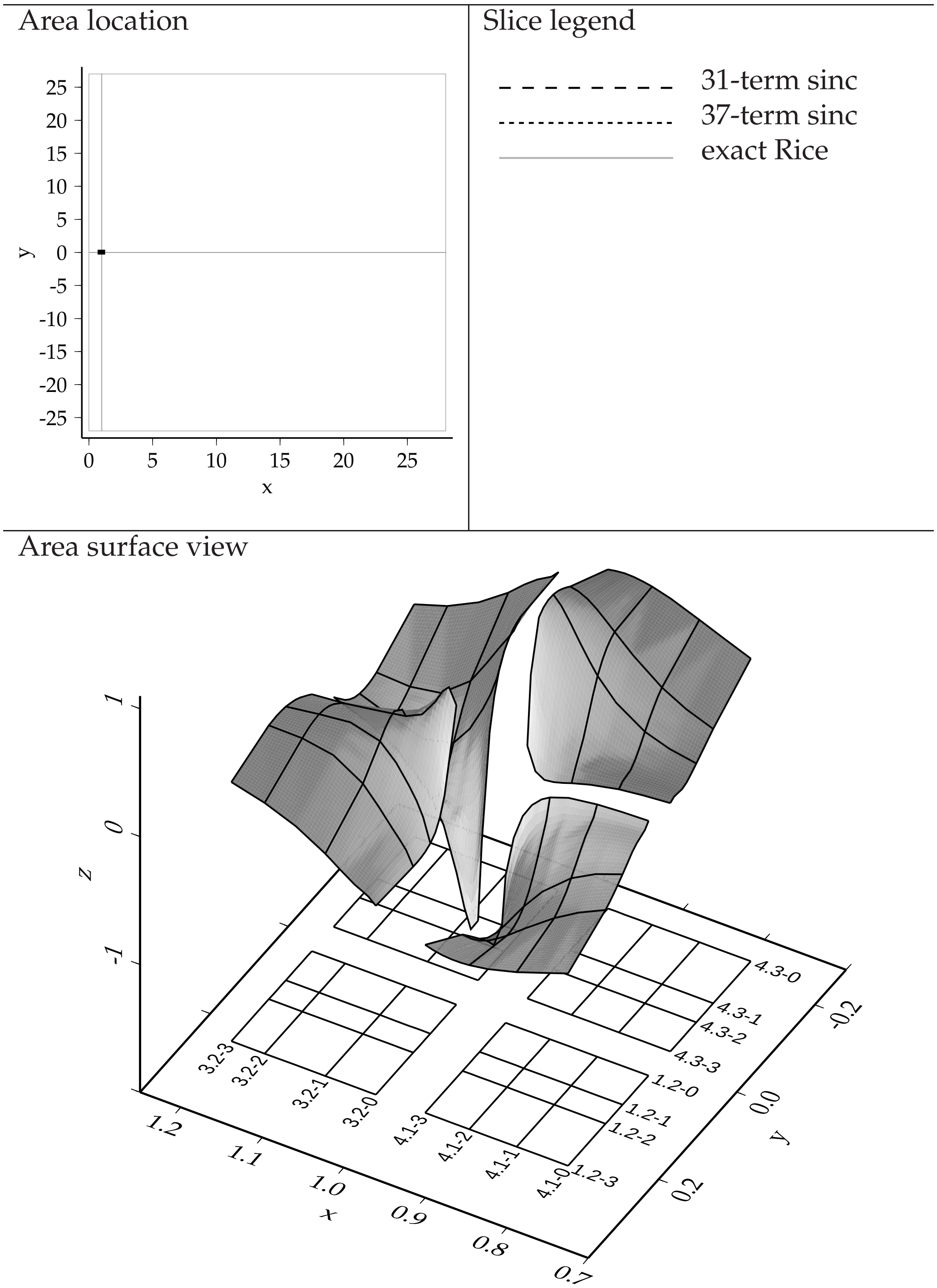


Figure 5.30. Graphs of τ^{12}, view 1, part 1: Location, legends and surface.

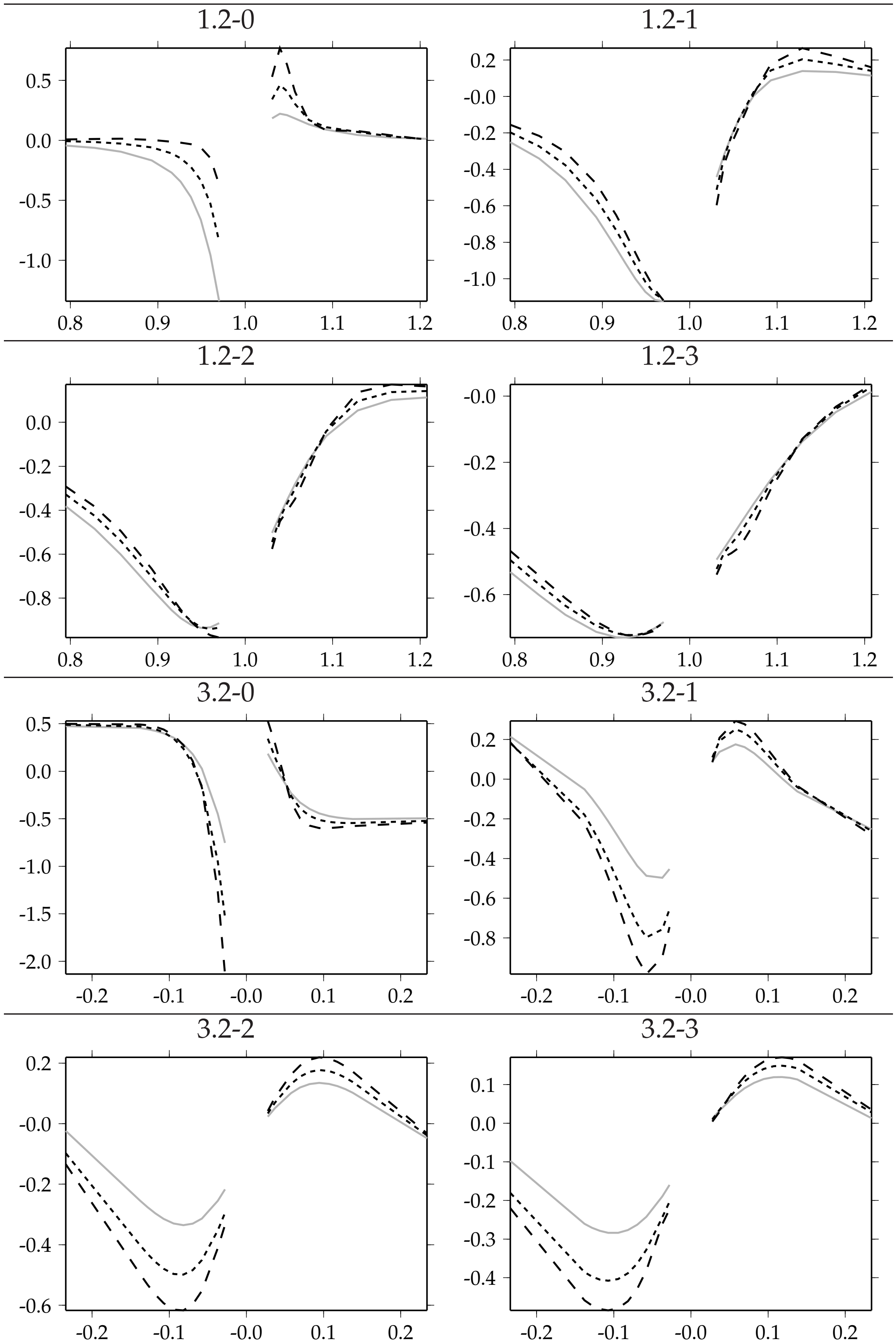


Figure 5.31. Graphs of τ^{12}, view 1, part 2: Detailed group slices.

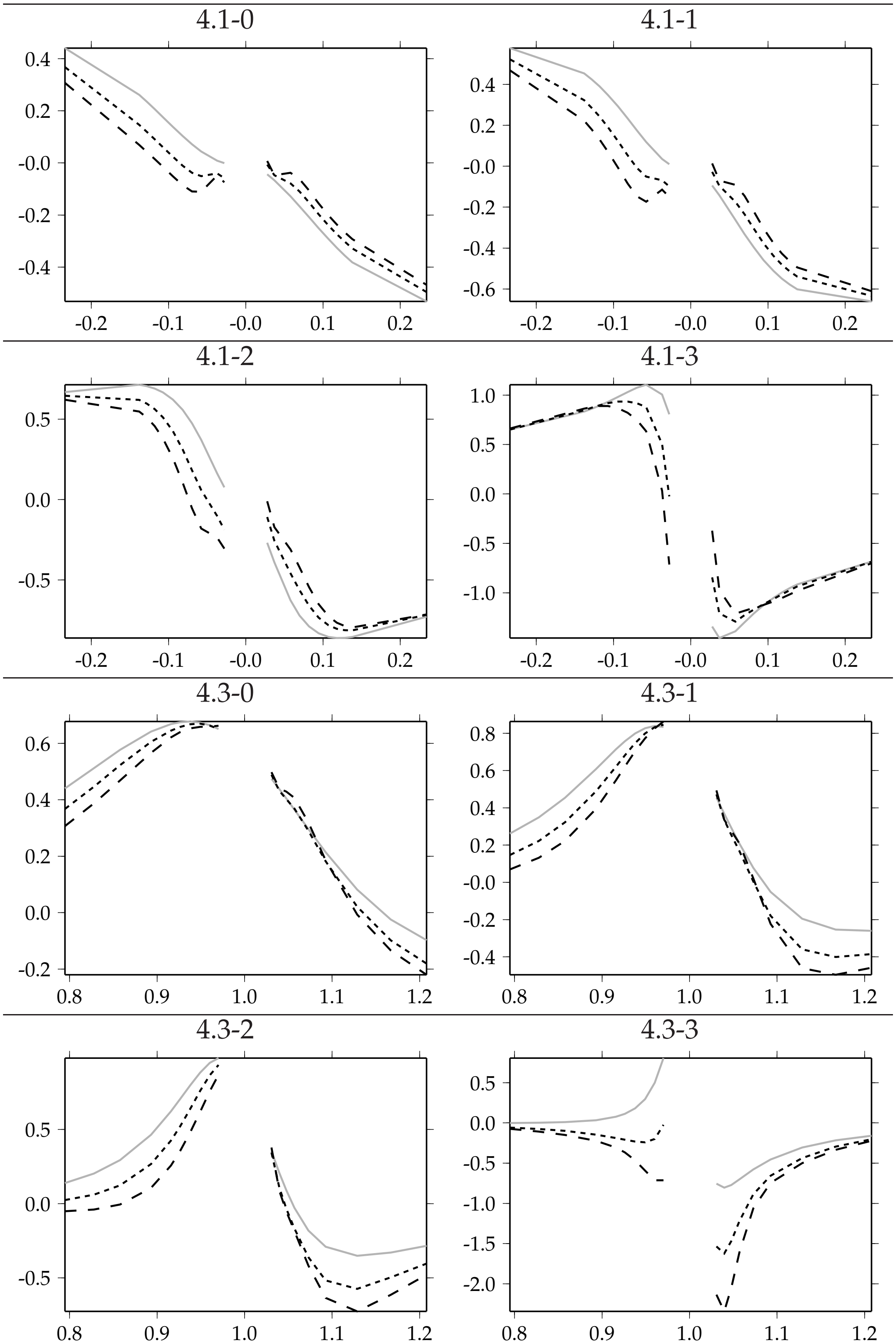


Figure 5.32. Graphs of τ^{12}, view 1, part 3: Detailed group slices, continued.

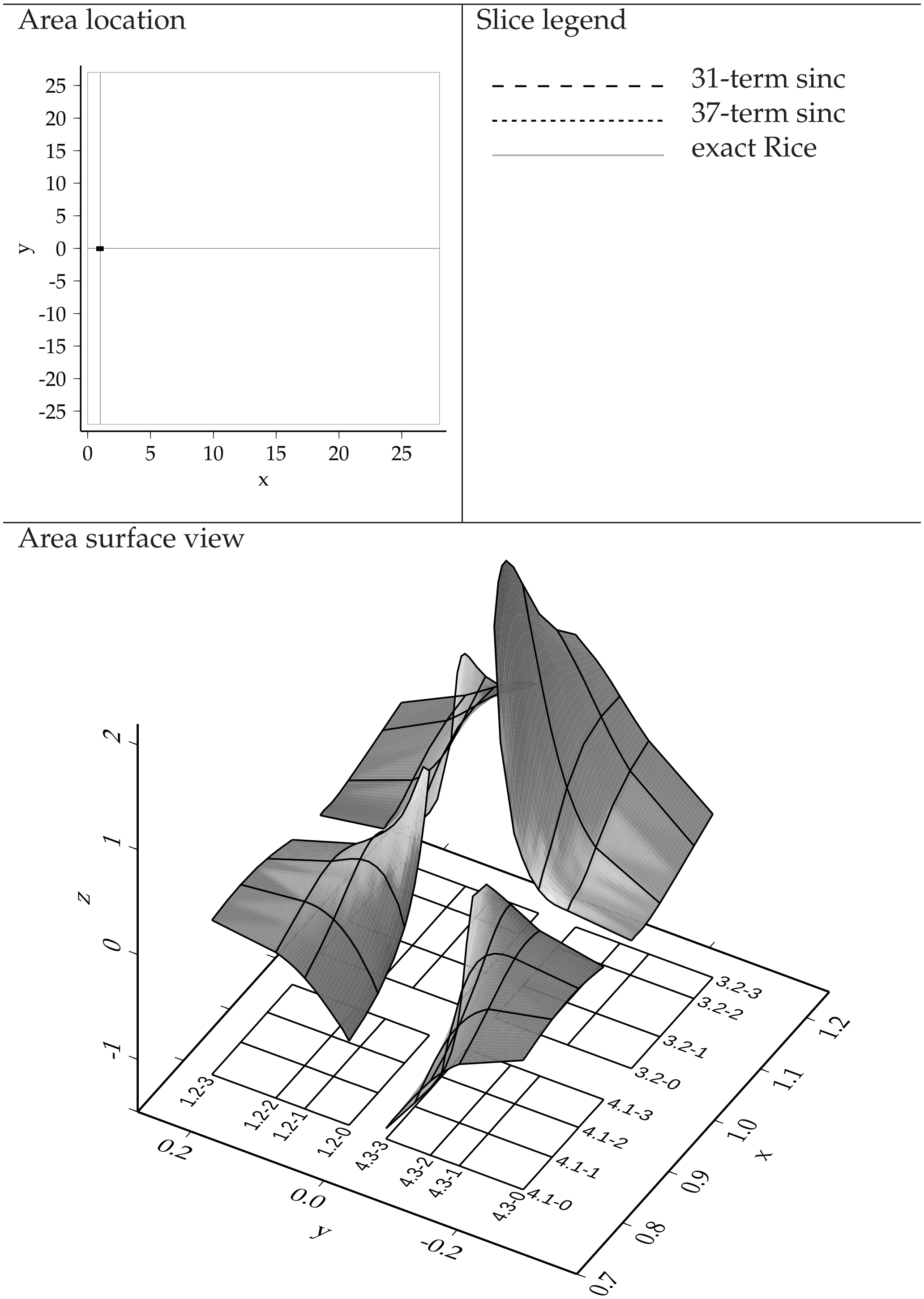


Figure 5.33. Graphs of τ^{11}, view 1, part 1: Location, legends and surface.

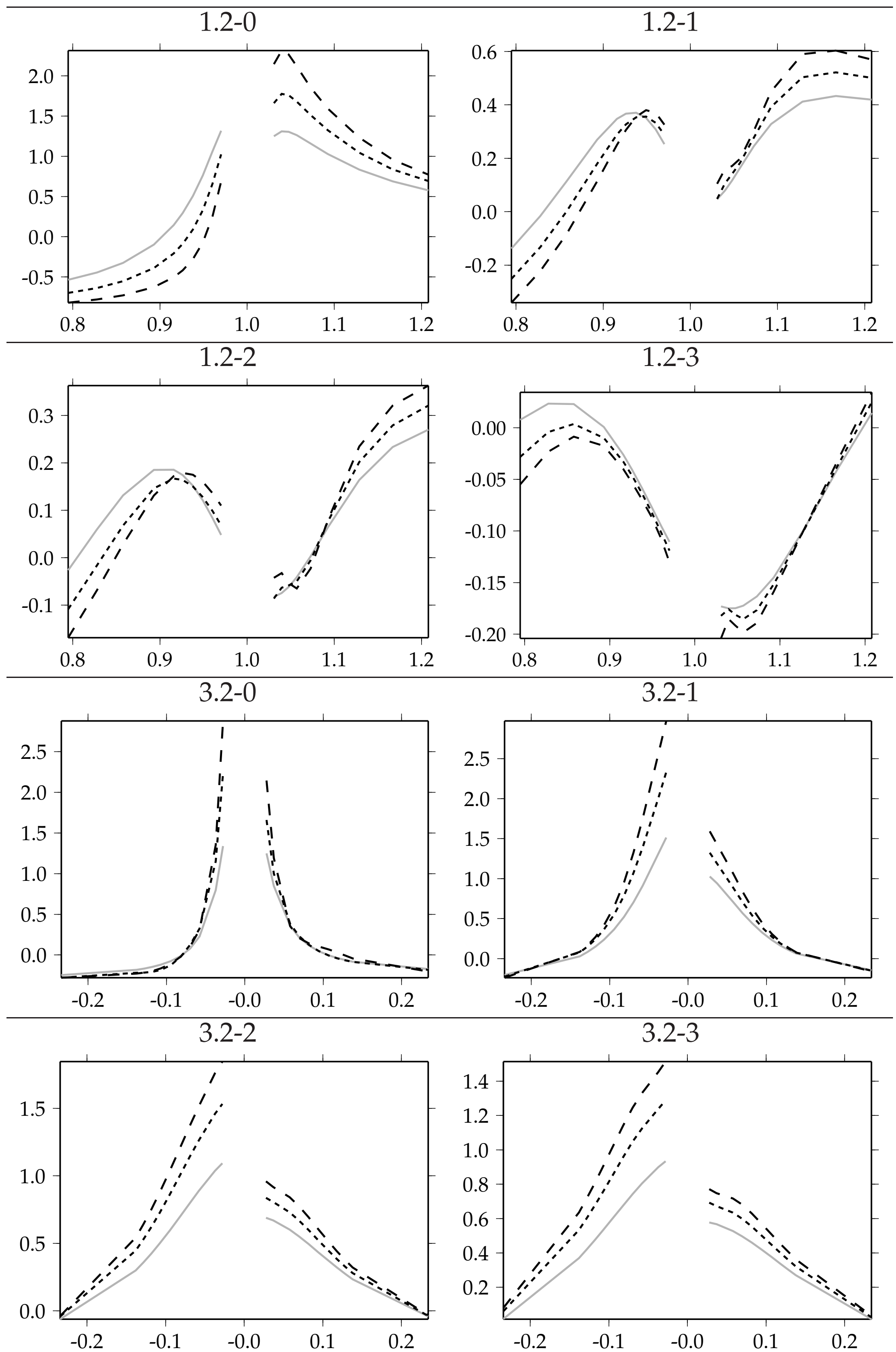


Figure 5.34. Graphs of τ^{11}, view 1, part 2: Detailed group slices.

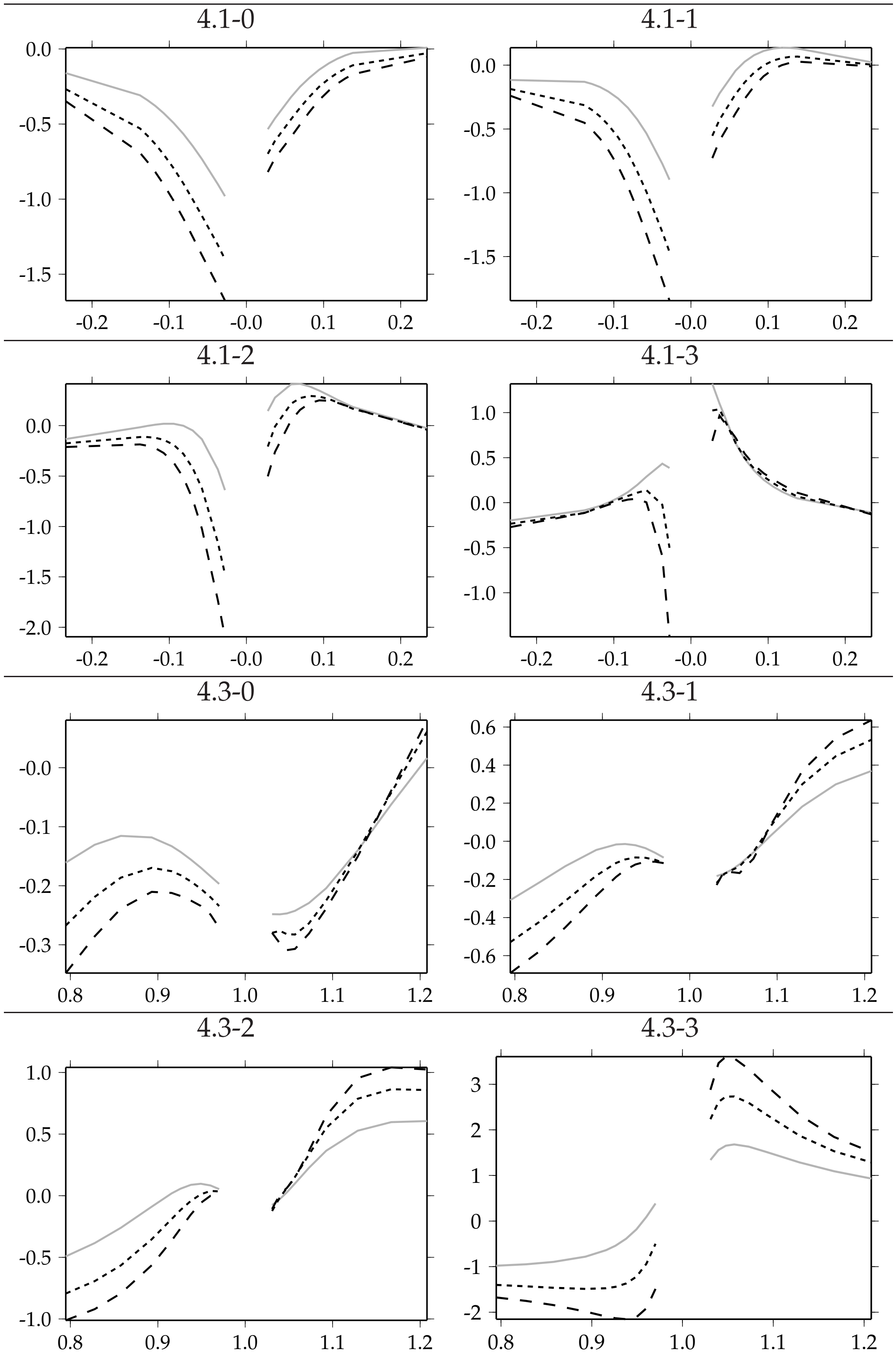


Figure 5.35. Graphs of τ^{11}, view 1, part 3: Detailed group slices, continued.

CHAPTER 6

ALGORITHM IMPLEMENTATION

A simple overview of the whole SINC-ELLPDE environment was given in Chapter 2. Now, consider again diagram 4.35. From an algorithmic point of view, the nodes correspond to data structures, while the edges correspond to algorithms.

For the mathematics, the equations and their properties are of paramount importance, while the transition from one stage to another (e.g., from the PDE system to the block system) is implicit, hence ignored. Thus, Chapter 4 provided this theoretical background, a mathematical notation for multiple-unknown multiple-rectangle systems, and mathematical illustrations of the PDE system, block system, discrete block system, discrete approximation and smooth approximation stages via several examples. Further, notation in Chapter 4 is intended for humans and is therefore ambiguous and highly context-dependent. Of course, such context is also implicit, usually requiring a thorough understanding of the ideas involved.

This chapter is concerned with the concrete, mechanical implementation of those mathematical ideas. In this chapter, as in the program it describes, the mathematical meaning of equations and their properties are now assumed in the programs, hence ignored, while the transitions from one stage to another (e.g., from the PDE system to the block system via the block conversion algorithm) are paramount. Thus, the focus now is on the algorithms effecting the transitions between data stages:

Block Conversion: This step takes systems of equations, written in the format specified by Figure 4.3 and produces the collection of block operators as illustrated in Equation 4.40.

Discretization: Given the block operators in program-accessible form, this step produces the discretized matrix blocks of Equation 4.53 (the constituents of $[L]$ and $[f]$) from the block operators, arranges them appropriately, and provides the full matrix $[L]$ and right-hand-side vector $[f]$.

Solution: Currently this is the most time-consuming of all steps, consisting of a simple brute-force program solving the linear system $[L][u] = [f]$ using the SUPERLU package described in (Demmel, Eisenstat, Gilbert, Li, and Liu 1999).

Reconstruction: The simple function of extracting the components $[u^{i|j}]$ from $[u]$ and constructing the unknown functions $u^{i|j}(x, y)$ via Equation 4.32 is handled here.

As these algorithms have very strict input and output format requirements, a secondary focus is on the files read and written by them. Additionally, notation now is meant to be machine-readable; it will therefore be exact, and the full context is always provided. As much as possible with these constraints, the notation has been kept human-readable.[1]

The related tasks of

- using computed unknowns for visualization and generation of human-readable graphs,
- automatic convergence checks,
- selection of proper data point positions,
- describing the programs used to verify the programs and algorithms presented here, and

[1] In fact, the mathematical notation of Chapter 4 was derived from the notation of this chapter, not vice versa. The implementation of an algorithm is everything — literally.

- developing a more efficient method of solving the linear system $[L][u] = [f]$

are left for future work.

While items here are dealt with in detail, conciseness of their representation varies significantly between programming languages. For the reasons detailed in the introduction, standard languages are severe hindrances in the implementation of complex algorithms. All novel parts of the present system are therefore implemented in the OCAML language.[2]

Various pieces of documentation are available for OCAML. An OCAML precursor, caml special light, is described (in French) in (Leroy 1995); a full programming book using OCAML was recently published, also in French: (Chailloux, Manoury, and Pagano 2000). The documentation provided with the (freely available) OCAML distribution (Leroy 1997) provides an overview and short tutorials for the major aspects of the OCAML system, as well as full library documentation.

The full algorithmic version of diagram 4.35 is shown in Figure 6.1. In the graph, files ending in .cmo and .cma are OCAML bytecode files produced from .ml source files, and files ending in .map use MAPLE-compatible nested list format. Other files' formats are mentioned with the explanations.

The correspondence between the diagram and the graph is described in detail in the following sections.

6.1 Block conversion

The PDE system is provided by the user in eqn_input.map and the block conversion is performed by pre_collocation, which generates the template data files T:dom_data and T:global_data, and the program files code.ml and data.ml.

---

[2] The SUPERLU library is used as is; it is written in Standard C.

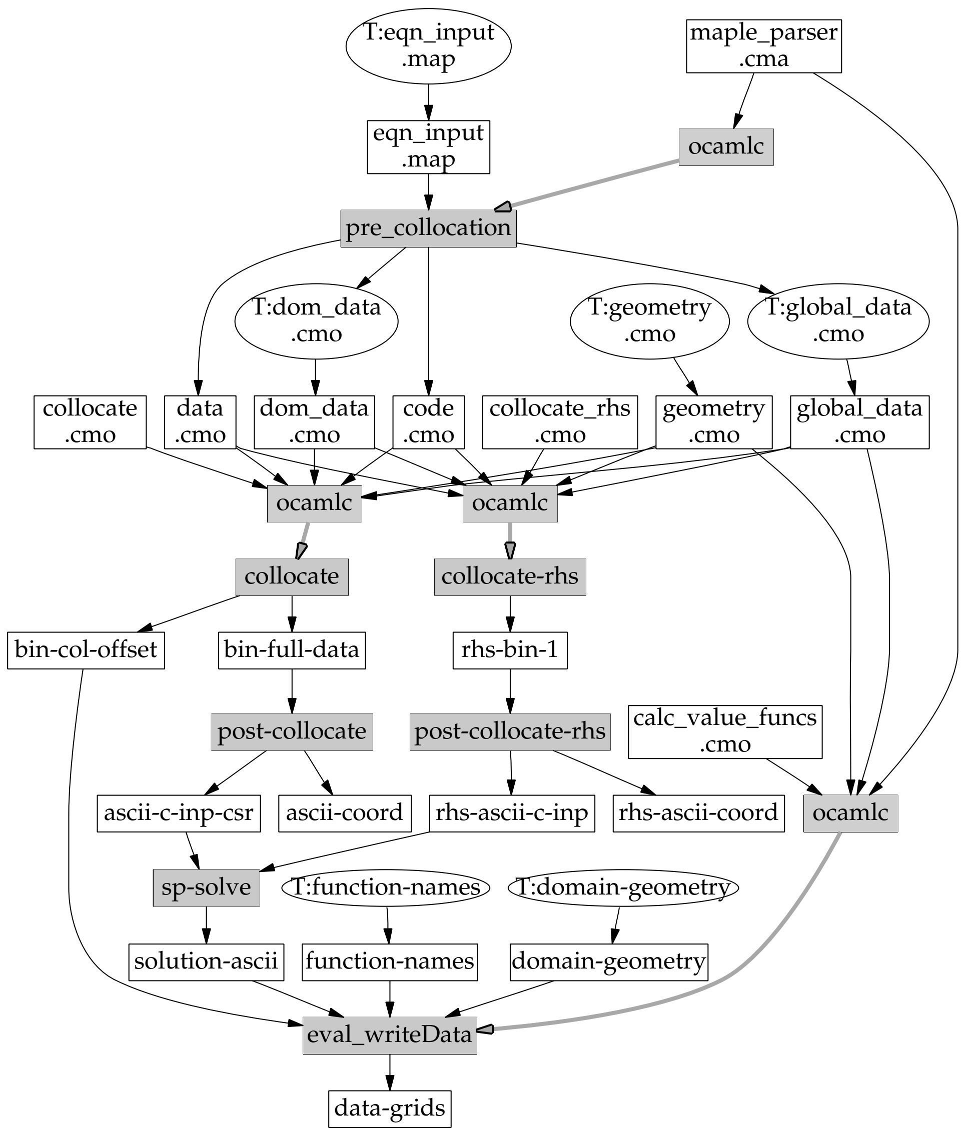


Figure 6.1. Data flow for input equation translation, matrix assembly/solution, and answer vector decomposition/solution evaluation. Elliptical data is user-provided, possibly by filling in a program-generated template. Shaded rectangles represent functions, plain rectangles data. Dark edges show data flowing to a program; light edges show programs creating programs.

6.1.1 Module eqn_input.map

The input file eqn_input.map resembles the typeset equations, e.g., pages 62–63; an example is shown in Figure 6.2, and the special operators are shown in Table 6.1.

6.1.2 Module pre_collocation

Further, the eqn_input.map file must conform to the BNF grammar of Figure 4.3. In pre_collocation, this is enforced partly by the lexer and parser, and completed by pattern matching. The LEX-compatible lexeme definitions are shown in Figure 6.3, the mappings for the special lexeme classes INFIXOP1 through INFIXOP5 are in Table 6.1, and the YACC-style parser in Figure 6.4. These are mostly standard definitions; the idea of keeping the lexer and parser small and flexible by using a multicharacter operator class is taken from OCAML. Also note that these grammatical definitions form a superset of MAPLE and MUPAD's expressions, facilitating production of equations by these systems.

Output for code.ml and data.ml is generated in two steps. Using the same tree traversal routine used to traverse the input expression, a simplified abstract tree representing the OCAML code is generated; this new tree is then converted to the pretty-printed text.

The template data file T:dom_data is created similarly: The input expression (in eqn_input.map) is scanned for variable names (e.g., nu), these are made part of the current rectangle's parameter list (e.g., Dom_1.nu), an intermediate tree is generated, and the intermediate tree is pretty-printed.

The template data file T:global_data is empty by default, and can be used to introduce arbitrary user-defined functions.

6.1.3 Modules code.ml and data.ml

code.ml contains the coefficient functions needed later by collocate; references to these are stored in a usable form in data.ml as a table[3] mapping (domain *

[3] Actual internal type is
`(domain * region * equation * side, code_linop_triple list) Hashtbl.t` Also there is

```
equation_specs=[
   unknowns=[u1, u2, u11, u12, u21, u22],
   domains=[
      1=[
         regions=[
            Interior=[
               ((-2*‘\nu‘+2)*~ u11\ 1+~ (1-2*‘\nu‘)*~ u12\ 2)+~ 1*~ u21\ 2=~0,
               ((1-2*‘\nu‘)*~ u21\ 1+~ (-2*‘\nu‘+2)*~ u22\ 2)+~ 1*~ u11\ 2=~0,
               1*~ u1\ 1+~ -1*~ u11=~0, 1*~ u1\ 2+~ -1*~ u12=~0,
               1*~ u2\ 1+~ -1*~ u21=~0, 1*~ u2\ 2+~ -1*~ u22=~0],
            Bottom=[
                -‘\mu‘*~ u1\ 2+~  -‘\mu‘*~ u2\ 1=~0,
               (((-2*‘\mu‘)*(-1+‘\nu‘))/(-1+2*‘\nu‘))*~ u2\ 2+~
               (((2*‘\mu‘)*‘\nu‘)/(-1+2*‘\nu‘))*~ u1\ 1=~‘\sigma‘,
               1*~ u1\ 1+~ -1*~ u11=~0, 1*~ u1\ 2+~ -1*~ u12=~0,
               1*~ u2\ 1+~ -1*~ u21=~0, 1*~ u2\ 2+~ -1*~ u22=~0],
            Left=[
               1*~ u21=~0, 1*~ u1=~0, 1*~ u1\ 1+~ -1*~ u11=~0,
               1*~ u1\ 2+~ -1*~ u12=~0, 1*~ u2\ 1+~ -1*~ u21=~0,
               1*~ u2\ 2+~ -1*~ u22=~0],
            Top=[
               1*~ u1=~0, 1*~ u2=~0, 1*~ u1\ 1+~ -1*~ u11=~0,
               1*~ u1\ 2+~ -1*~ u12=~0, 1*~ u2\ 1+~ -1*~ u21=~0,
               1*~ u2\ 2+~ -1*~ u22=~0],
            Right=[
               1*~ u11=~0, 1*~ u21=~0, 1*~ u1\ 1+~ -1*~ u11=~0,
               1*~ u1\ 2+~ -1*~ u12=~0, 1*~ u2\ 1+~ -1*~ u21=~0,
               1*~ u2\ 2+~ -1*~ u22=~0], RightOL=[1*~ u1=~0, 1*~ u2=~0]]],
      2=[
         regions=[
            LeftOL=[-1*~ u11=~0, -1*~ u21=~0],
            ...]]]]
```

Figure 6.2. Parts of the machine-readable form of the single-material equations on pages 62–63. This input file corresponds to Figure 6.1, node eqn_input.map. The special operator's (+~, etc.) analogues are shown in Table 6.1.

Table 6.1. The connections between the mathematical infix notation, its program-readable infix and prefix versions, and the internal representation. Together with the lexer in Figure 6.3, the parser of Figure 6.4 and a pattern matcher, this forms the input reading and verification portion for Figure 6.1, node pre_collocation.

Infix Math Form	Infix Program Form	Prefix Name	Internal Form
$\leq$	<=	LEQ	LEQ
$<$	<	Less	Less
$\geq$	>=	GEQ	GEQ
$>$	>	Greater	Greater
$=$	=	Equal	Equal
$\neq$	<>	NEQ	NEQ
$+$	+	Plus	Plus
$-$	-	Minus	Minus
$\cdot$	*	Times	Times
$/$	/	Divide	Divide
	**	Power	Power
	^	Power	Power
$\cong$	=~	OpEqn	OpEqn
$\oplus$	+~	OpSum	OpSum
$\ominus$	-~	OpMinus	OpMinus
$\odot$	*~	OpProd	OpProd
$\backslash$	\	OpDiff	OpDiff

```
rule token = parse
  | " "
  | "\t"
  | "\n"                               (* skip blanks *)
  | ["=" "<" ">" ]+ ["~" "^" "#"]*     { INFIXOP1 }
  | ["+" "-"]["~" "^" "#"]*            { INFIXOP2 }
  | "\\" ["~" "^" "#"]*                { INFIXOP5 }
  | "**" ["~" "^" "#"]*                { INFIXOP4 }
  | "^" ["~" "^" "#"]*                 { INFIXOP4 }
  | ["*" "/"]["~" "^" "#"]*            { INFIXOP3 }
  | ["0"-"9"]* ("." ["0"-"9"]*)
    (["e" "E"] ["+" "-"]? ["0"-"9"]+)?  { FLOAT  }
  | ["0"-"9"]+                         { INT }
  | "`"                                { SYMBOL  }
  | "\""                               { STRING  }
  | "("                                { LPAREN }
  | ")"                                { RPAREN }
  | "["                                { LBRACKET }
  | "]"                                { RBRACKET }
  | "{"                                { LBRACE }
  | "}"                                { RBRACE }
  | ","                                { COMMA }
  | ["a"-"z" "A"-"Z" "_"]
    ["a"-"z" "A"-"Z" "_" "0"-"9"]*     { SYMBOL }
  | "!"
    ["a"-"z" "A"-"Z" "_" "0"-"9"]+     { MATCH_SYM }
  | eof                                { EOF }
  | _                                  { Illegal character }
and string = parse
  | "``"                               { constituent }
  | "`" { finish }
  | "\\" ("\010" | "\013" | "\013\010")
    [" " "\009"] *                     { constituent }
  | "\\" ["\\" """ "n" "t" "b" "r"]    { constituent }
  | eof                                { error }
  | "\n"                               { constituent }
  | _                                  { constituent }

and real_string = parse
  | "\"\""                             { constituent }
  | """                                { finish }
  | "\\" ("\010" | "\013" | "\013\010")
    [" " "\009"] *                     { constituent }
  | "\\" ["\\" """ "n" "t" "b" "r"]    { constituent }
  | eof                                { error }
  | "\n"                               { constituent }
  | _                                  { constituent }
```

Figure 6.3. The ocamllex lexer portion of the program implementing the BNF grammar of Figure 4.3. There are three types of lexeme, token, string and real_string; the analyzer for each starts with the parse keyword. The attached actions are simplified here for illustration. Together with the special lexemes of Table 6.1 (for recognized INFIXOPs and functions), the parser of Figure 6.4 and a pattern matcher, this forms the input reading and verification portion for Figure 6.1, node pre_collocation. The full program is 219 lines.

```
(* Precedences and associativities. Lower precedences come first. *)
%left     COMMA                (* sequences *)
%left     INFIXOP1             (* = < > etc *)
%left     INFIXOP2             (* + - *)
%left     INFIXOP3             (* * / *)
%nonassoc UMINUS               (* unary - *)
%right    INFIXOP4             (* **, ^ *)
%left     INFIXOP5             (* \ *)
%nonassoc LPAREN LBRACKET LBRACE
%nonassoc CALL                 (* function call, structure index *)
%%
main:
      expr_list EOF               { (rev $1) }
      ;
expr:
      INT                         { Int($1) }
    | FLOAT                       { Float( $1) }
    | STRING                      { String($1) }
    | SYMBOL                      { Symbol($1) }
    | MATCH_SYM                   { Match_sym($1) }
    | LBRACKET expr_list RBRACKET { List (rev $2) }
    | LBRACE expr_list RBRACE     { Set (rev $2) }
    | LPAREN expr RPAREN          { $2 }
    | expr INFIXOP1 expr          { binary_oper $2 ($1 , $3) }
    | expr INFIXOP2 expr          { binary_oper $2 ($1 , $3) }
    | expr INFIXOP3 expr          { binary_oper $2 ($1 , $3) }
    | expr INFIXOP4 expr          { binary_oper $2 ($1 , $3) }
    | expr INFIXOP5 expr          { binary_oper $2 ($1 , $3) }
    | INFIXOP2 expr %prec UMINUS  { unary_oper $1 $2 }
    | expr LPAREN expr_list RPAREN %prec CALL { handle_function }
    | expr LBRACKET expr_list RBRACKET %prec CALL { Indexed }
    ;
expr_list:                        { [] }
    | expr                        { [$1] }
    | expr_list COMMA expr        { $3::$1 }
    ;
%%
```

Figure 6.4. The ocamlyacc expression parser of the program implementing the BNF grammar of Figure 4.3. Calls to binary_oper map the recognized infix symbols of Table 6.1 to their internal form; calls to handle_function map the recognized functions. Together with the lexer of Figure 6.3, the special lexemes of Table 6.1 and a pattern matcher, this forms the input reading and verification portion for Figure 6.1, node pre_collocation. The full program is 165 lines; here, only the simplified rules are shown.

region * equation * side) to ((function * unknown * operator) list).

The list corresponds to a list of linear operator expressions of the form

$$f(x,y)\mathrm{Op}(u)(x,y) \tag{6.1}$$

and is later used for block construction.

See Figures 6.5 and 6.6 for samples of these generated files.

6.1.4 Modules dom_data.ml and global_data.ml

dom_data.ml contains problem-specific user-provided numerical values; after selecting appropriate values, it has the form of Figure 6.7.

The template data file T:global_data is empty by default, and can be used to introduce arbitrary user-defined functions.

6.2 Discretization

The production of $[L]$ and $[f]$ proceeds in three steps. First, the geometry and sinc parameters are provided by the user in the file geometry. These are combined by ocamlc, with the previously generated files code.ml and data.ml, the user-filled template files dom_data.ml and global_data.ml, and the main part of the matrix discretization algorithm provided in collocate.ml and the right-hand-side vector discretization algorithm in collocate_rhs.ml, to form the collocate and collocate-rhs programs. Second, the program collocate is run to produce the matrix blocks constituting $[L]$ while collocate-rhs is run to produce the vector blocks constituting $[f]$. The matrix blocks are stored in portable binary format in bin-full-data; the vector blocks are stored in rhs-bin-1. The starting indices of the unknowns' coefficients in the vector $[u]$ (which are column offsets in $[L]$) are stored in bin-col-offset for later use in solution reconstruction. Third, multiple values for given matrix/vector entries are resolved by post-collocate/post-collocate-rhs, and the full matrix is made available in the compressed sparse row (CSR) format file ascii-c-inp-csr, and a coordinate format file ascii-coord. The right-hand-side vector

no real need in OCAML to provide functions via top-level definitions; a single file would have sufficed. This is a good example of the handicaps of FORTRAN, C, C++ and JAVA.

```
(* Header.*)
open Coll_gen_dom_data ;;
(* Generated code. *)

let _gensym_f1 x y =  1.0
...
let _gensym_f127 x y =
    ((2.0*. (!Dom_2._mu))*. (!Dom_2._nu))/.
    (-1.0+. 2.0*. (!Dom_2._nu))
...
let _gensym_f368 x y =  0.0
```

Figure 6.5. Generated Functions implementing the coefficients of linear operators, corresponding to Figure 6.1, node code.cmo. Total length is 385 lines; most deleted entries return the values 0.0, -1.0 or 1.0. The Dom_1 etc. parameter modules are defined in domain_data.cmo.

```
(* Header.*)
open Collocation_types;;
let code_str = empty_code_struct() ;;
let ins key value = Hashtbl.add code_str.cd_dom_table key value ;;
(* Generated code. *)

let _ = ins (1, Top, 1, Lhs) [
        { cd_f = Coll_gen_code._gensym_f1 ; cd_op = I ;
          cd_u = "u1" }];;
let _ = ins (1, Top, 1, Rhs) [
        { cd_f = Coll_gen_code._gensym_f2 ; cd_op = I ;
          cd_u = "" }];;
...
let _ = ins (4, RightOL, 2, Lhs) [
        { cd_f = Coll_gen_code._gensym_f367 ; cd_op = I ;
          cd_u = "u2" }];;
let _ = ins (4, RightOL, 2, Rhs) [
        { cd_f = Coll_gen_code._gensym_f368 ; cd_op = I ;
          cd_u = "" }];;
```

Figure 6.6. Generated functions/operator table providing the coefficients of the linear operators to collocate. This listing corresponds to Figure 6.1, node data.cmo. Total length is 639 lines. _gensym_* references are defined by code.ml; the ins function adds entries to the (domain * region * equation * side) $\rightarrow$ ((function * unknown * operator) list) table used by collocate.

```
module Dom_1 = struct
     let _nu = ref 0.33
     let _sigma = ref 1.00
     let _mu = ref 1.0
     end
module Dom_2 = struct
     let _nu = ref 0.33
     let _sigma = ref 1.00
     let _mu = ref 1.0
     end

module Dom_3 = struct
     let _nu = ref 0.33
     let _sigma = ref 1.00
     let _mu = ref 3.21
     end
module Dom_4 = struct
     let _nu = ref 0.33
     let _sigma = ref 1.00
     let _mu = ref 3.21
     end

let domainNames = [
     (1, ["_nu"; "_sigma"; "_mu"]) ;
     (2, ["_sigma"; "_mu"; "_nu"]) ;
     (3, ["_mu"; "_nu"; "_sigma"]) ;
     (4, ["_nu"; "_sigma"; "_mu"]) ]
```

Figure 6.7. Rectangle- and problem-specific numerical constants, corresponding to Figure 6.1, node dom_data.cmo.

is made available as simple vector in rhs-ascii-c-inp, and using coordinate format in rhs-ascii-coord.

6.2.1 Module geometry

The geometry file contains problem geometry and sinc numerical parameters; a sample is shown in Figure 6.8.

6.2.2 Modules collocate.ml and collocate-rhs.ml

In this section, the algorithm generating the discrete sparse matrix blocks and their alignment is described. The calculation of the rhs vector (shown by the -rhs nodes) is similar and simpler, and will not be described.

The calculation of any single entry is shown in Figure 6.9.

The functions using these data could be "pure", i.e., they would have no state, take only the indicated argument (anonymously — without requiring that the value has the global name shown), and return only an anonymous instance of the indicated type. Unfortunately this is only the first of two steps, and the second step, accumulation of these entries into a data structure, was done in a fully imperative style, as shown in Figure 6.10.[4]

The result of accumulating the individual entries is a table with four indices, each entry of which is a sparse matrix block in CSR[5] format and its associated offset in the full matrix. More precisely, the result is a structure of type

```
val gBlocks_table :
  (domain * region * equation * unknown,
   column_block list)
  Hashtbl.t ref
```

with subtypes

```
type region =
    Top | Left | Bottom | Right | Interior |
    TopOL | LeftOL | BottomOL | RightOL  and
    equation = int and
```

[4] This greatly complicated existing routines and data flow, and was a poor choice; a simpler functional version may be written in the future.

[5] Compressed Sparse Row.

```
let pi = 3.1415926535897932385
let po2 = pi/.2.0

let dom_num_grid = [|           (* Domain numbers, starting from 1. *)
  [| 1; 2 |] ;
  [| 4; 3 |]
|]
and dom_x_bounds =              (* Positions of the boundaries.*)
  [| 0.0 ; 1.0 ; 28.0 |]
and dom_y_bounds = [|
  27.0 ;
  0.0 ;
  -27.0
|]
and dom_x_sinc_alpha =
  [| 1.0 ; 1.0 |]
and dom_y_sinc_alpha = [|
  1.0 ;
  1.0
|]
and dom_x_sinc_d =
  [| po2 *. 1.19 ; po2 *. 1.19 |]
and dom_y_sinc_d = [|
  po2 *. 1.19;
  po2 *. 1.19
|]
and dom_x_terms =               (* Series summation limits. *)
  [| (3, 3); (3, 3) |]
and dom_y_terms = [|
  (3, 3) ;
  (3, 3)
|]
and gTrue_unknowns = ["u1"; "u2"] and
    g1st_order_sys_unknowns = ["u11"; "u12"; "u21"; "u22"]
```

Figure 6.8. Geometry and sinc numerical parameters for sinc core. These data correspond to Figure 6.1, node geometry.cmo.

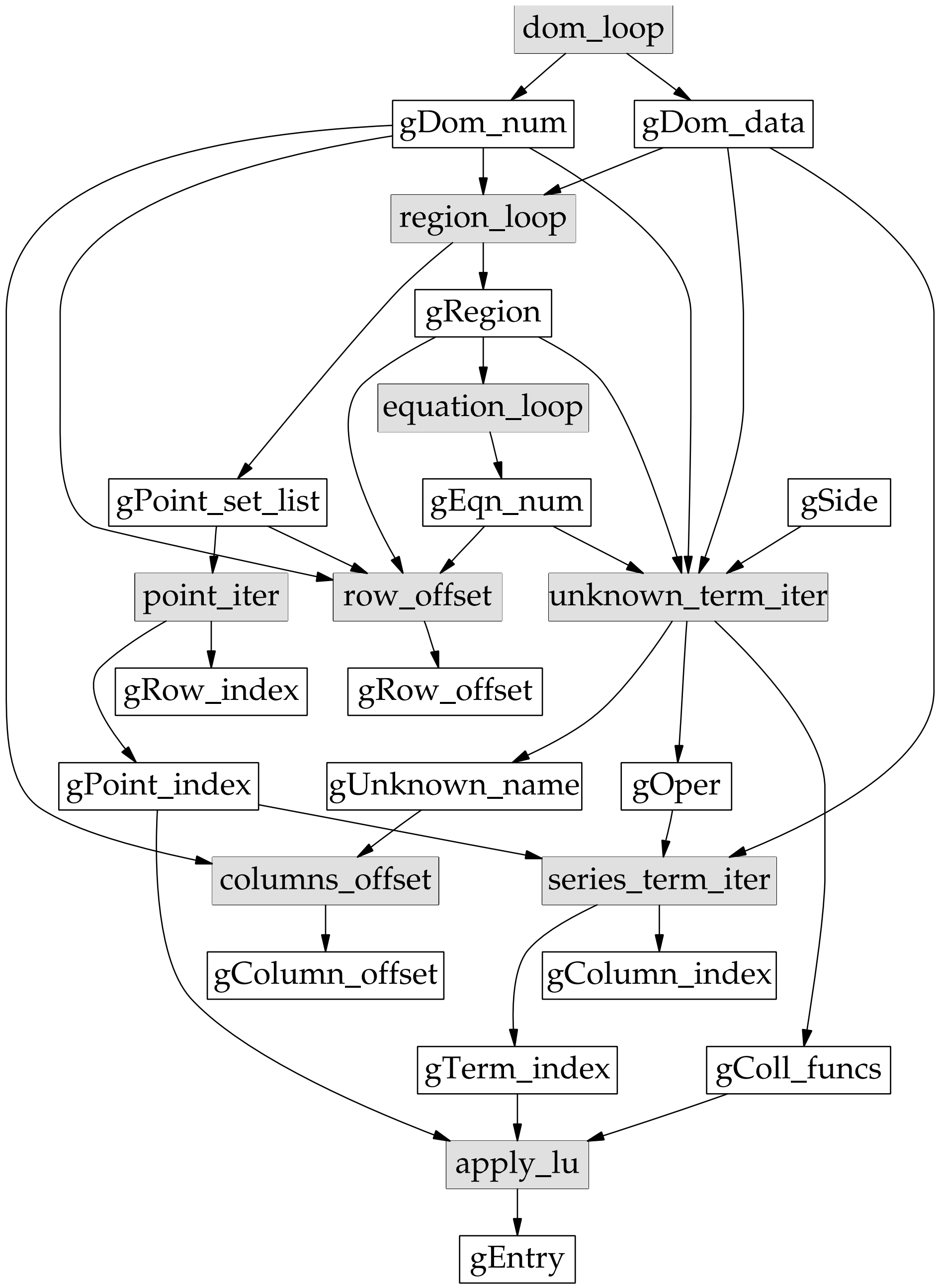


Figure 6.9. Calculation of any single entry of the collocation matrix. This graph shows the core idea of Figure 6.1, node collocate. The shaded rectangles are functions, the plain rectangles data; solid lines indicate data creation.

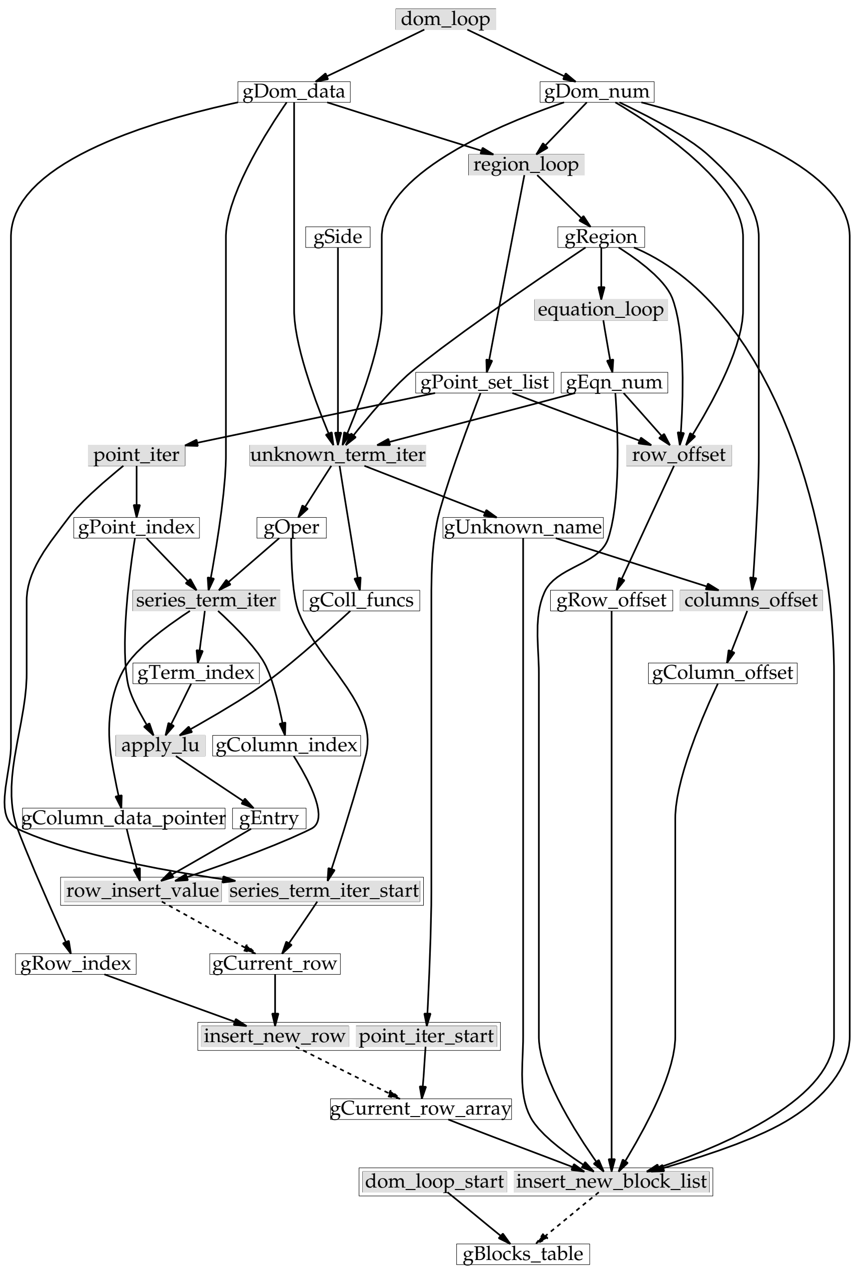


Figure 6.10. Calculation and accumulation of all collocation matrix entries; this graph shows the full internals of Figure 6.1, node collocate. The shaded rectangles are functions, the plain rectangles data; solid lines indicate data creation, and dashed lines are modifications/additions to existing state. The module implementing this is 682 lines; the type definitions and utility functions take up another 293 lines.

```
domain = int and
unknown = string and
column_block = { column_offset : int ;
                 row_offset : int ;
                 csr_block : mat_entry array array}
```

and

```
type mat_entry = { column_index : int ; entry : float }
```

Most of these matrix blocks have correct positions; the overlap blocks' positions are adjusted later by collocate.

The addition of value accumulation to Figure 6.9 results in Figure 6.10. Functions of the same name in Figure 6.10 are longer, expanded versions of those in Figure 6.9, while data of the same name remains unchanged. The functions in Figure 6.10 were implemented in a simple imperative style: they read module-level variables, and write to module-level variables. The functions themselves thus have type `unit -> unit`.

Having shown the data flow in Figure 6.10, a figure illustrating control flow (function calling order) is helpful. To avoid further complexity in existing figures[6], the control flow is shown separately in Figure 6.11.

The functions in these figures are all implemented in `collocate.ml`; because of their complexity, a more detailed explanation is in order. Following are descriptions of the types of the data, together with short explanations of the functions of Figure 6.10; this should give sufficient detail to allow independent implementation. Purposely omitted here are dependencies on core sinc functionality, data obtained from other modules previously described, and utility functions from the other modules.

It should be noted that `gSide` has type `type side = Lhs | Rhs` and its value is always `Lhs` for `collocate`, and `Rhs` for `collocate-rhs`.

6.2.2.1 Function dom_loop_start

A trivial function meant to clarify program flow, `dom_loop_start` initializes the `gBlocks_table` and calls `dom_loop`.

[6] The direct addition of control flow information to Figure 6.10 results in an unreadable graph.

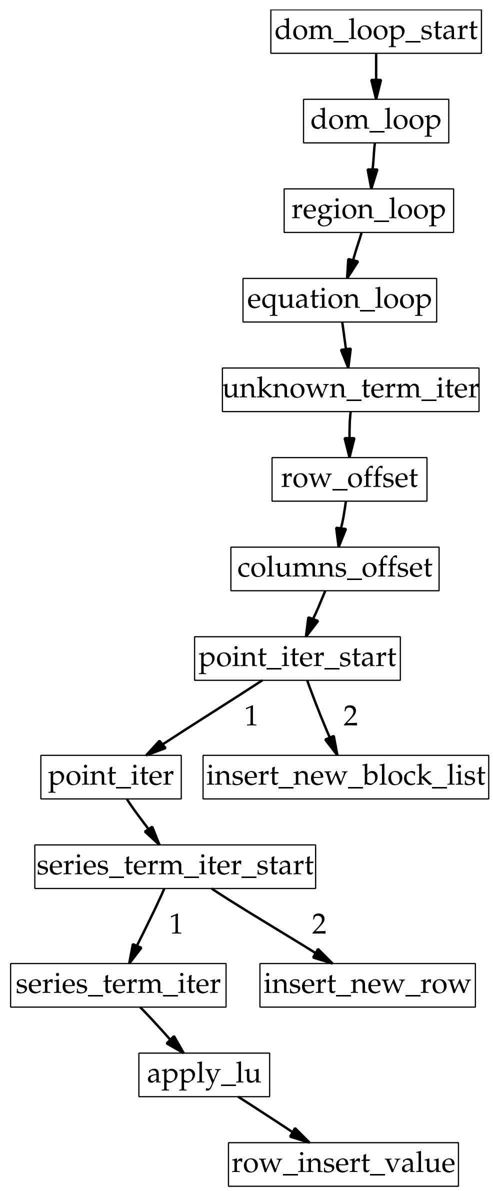


Figure 6.11. Nested function call sequence for calculation and accumulation of all collocation matrix entries.

6.2.2.2 Function dom_loop

This loop iterates over all domains (rectangles) in the `dom_num_grid` array of `collocate_geometry.ml`; for every domain it sets the values

```
val gDom_data : dom_data ref
```

and

```
val gDom_num : domain ref,
```

of types

```
type dom_data =
  { a_1: float;
    b_1: float;
    m_1: int;
    n_1: int;
    a_2: float;
    b_2: float;
    m_2: int;
    n_2: int;
    alpha_1: float;
    d_1: float;
    alpha_2: float;
    d_2: float }
```

and

```
type domain = int,
```

respectively. It then calls `region_loop` to continue.

6.2.2.3 Function region_loop

This loop iterates over all regions, and for those relevant to the current domain, produces

```
val gRegion : region ref
```

and

```
val gPoint_set_list : point_set list ref;
```

the types in full are

```
type point_set =
  { x_min_index: int;
    x_max_index: int;
    y_min_index: int;
    y_max_index: int }
```

and

```
type region =
    Top
  | Left
  | Bottom
  | Right
  | Interior
  | TopOL
  | LeftOL
  | BottomOL
  | RightOL
```

respectively.

It then calls `equation_loop`.

6.2.2.4 Function equation_loop

This loop iterates over all equations in the current region, setting `val gEqn_num : equation ref` and calling `unknown_term_iter` each time. Equations are described by `type equation = int`.

6.2.2.5 Function unknown_term_iter

Using the `gDom_num`, `gRegion`, `gEqn_num` and `Lhs` as indices, `unknown_term_iter` retrieves the list of (coefficient, operator, unknown) terms provided by `pre_collocation`, and for every term, sets `gOper`, `gUnknown_name` and `gColl_funcs`, and calls `row_offset`.

The list of coefficients provided by `pre_collocation` is accessed via the helper function

```
val get_triple_list :
  domain * region * equation * side -> code_linop_triple list
```

The first two types are trivial:

```
val gOper : operator ref
```

with

```
type operator = I | D1 | D2,
```

and

```
val gUnknown_name : unknown ref
```

with

```
type unknown = string,
```

while the third type is more interesting:

```
val gColl_funcs : coll_funcs ref
```

of

```
type coll_funcs =
  { lu: int ->
        int -> float -> float -> int -> int -> float;
    x_j: int -> float;
    y_k: int -> float }
```

In `type coll_funcs`, `lu` is a closure providing

$$\mathrm{lu}(i,j,x,y,k,l) \;= f(x,y)\mathrm{Op}\left(\gamma_i(x)\gamma_j(y)\right)(x_k,y_l) \tag{6.2}$$

where Op is one of $\{I, \partial x, \partial y\}$, x_k is sinc point k in the x-direction, and y_l is sinc point l in the y-direction. Also provided are the functions `x_j` and `y_k` which map the integer indices of the sinc collocation points to the actual point positions.

6.2.2.6 Function row_offset

This function's sole purpose is to provide the appropriate row-index offset for the current domain, region, and equation, and continue by calling `columns_offset`. It provides the offset in `gRow_offset`. As may be expected, the offset is a simple integer: `val gRow_offset : int ref`.

6.2.2.7 Function columns_offset

Another simple function, `columns_offset` provides the starting column appropriate to the current domain and unknown in `gColumn_offset`; this again has the simple type `val gColumn_offset : int ref`. Control then passes to `point_iter_start`.

6.2.2.8 Function point_iter_start

This function is the first to serve two purposes, the calculation of values, and their accumulation. To this end, the `gCurrent_row_array` is initialized, and filled via the call to `point_iter`; this array is then stored in the `gBlocks_table` by a call to `insert_new_block_list`.

The types are

```
val gCurrent_row_array :  mat_entry array array ref
```

and

```
type mat_entry = { column_index: int; entry: float };
```

together, these form CSR blocks.

6.2.2.9 Function point_iter

For every point in the current region's point set, this function sets the `gPoint_index` and `gRow_index` and calls `series_term_iter_start`. The values set are simple,

```
val gPoint_index : point_index ref
```

with

```
type point_index = { x_ind: int; y_ind: int }
```

and `val gRow_index : int ref`. This function corresponds to the (point) τ mapping mentioned in the method chapter, Section 4.5.3, and provides the mapping of two-dimensional index pairs to a one-dimensional vector $(j, k) \to l$.

6.2.2.10 Function series_term_iter_start

Another dual-purpose function, `series_term_iter_start` sets up the `gCurrent_row`, fills it via a call to `series_term_iter`, and stores the accumulated values via `insert_new_row`.

The value

```
val gCurrent_row : mat_entry array ref
```

holds a single compressed row vector.

6.2.2.11 Function series_term_iter

Every unknown is represented by the series

$$\sum_i \sum_j c_{ij} \gamma_i(x) \gamma_j(y) \tag{6.3}$$

Using `gDom_data` to determine limits for i and j and `gPoint_index` to determine the current position (x, y), `series_term_iter` sets proper values for `gTerm_index`, `gColumn_index`, and `gColumn_data_pointer` using the current `gOper`, and calls `apply_lu` to calculate the current entry's value.

The value

```
val gTerm_index : term_index ref
```

with

```
type term_index = { serInd_1: int; serInd_2: int }
```

provides the (i, j) indices for the series, while `val gColumn_index : int ref` provides the single column index for the current unknown; it is used with `gColumn_offset` to get the proper final index into the matrix $[L]$.

Lastly,

```
val gColumn_data_pointer : int ref
```

provides the value's proper index for insertion into `gCurrent_row`.

6.2.2.12 Function apply_lu

Using `gTerm_index` `gPoint_index` and `gColl_funcs`, set the value of `gEntry` and continue with `row_insert_value`.

Of course, the type is `val gEntry : float ref`.

6.2.2.13 Function row_insert_value

Using `gColumn_data_pointer` and `gColumn_index`, add the current `gEntry` to `gCurrent_row`.

6.2.2.14 Function insert_new_row

Another trivial function used to clarify the program, `insert_new_row` inserts the `gCurrent_row` into the `gCurrent_row_array` at the index `gRow_index`.

6.2.2.15 Function insert_new_block_list

This function constructs a `column_block` using the now completed `gCurrent_row_array`, `gRow_offset` and `gColumn_offset`, and inserts the `column_block` into the `gBlocks_table`, under the index `(gDom_num, gRegion, gEqn_num, gUnknown_name)`.

The types are

```
val gBlocks_table :
    (domain * region * equation * unknown, column_block list)
    Hashtbl.t ref
```

with

```
type column_block =
  { column_offset: int;
    row_offset: int;
    csr_block: mat_entry array array }
```

and

```
   val gRow_index : int ref
```

6.2.3 Module bin-col-offset

Every unknown is represented by a series

$$\sum_i \sum_j c_{ij}\gamma_i(x)\gamma_j(y); \tag{6.4}$$

all of these c_{ij} are contained in $[u]$, which is eventually stored in `solution-ascii`.

This file contains the starting positions of c_{ij} for every unknown in every domain, in the form of a table mapping a given (domain, unknown) pair to the starting index of the corresponding coefficient subvector. The datum thus has type `(domain * unknown, int) Hashtbl.t`.

6.2.4 Module post-collocate

post-collocate serves two functions. First, there may be multiple values for a given matrix entry. These arise when an unknown appears multiple times in an `(function * unknown * operator) list`, and are always added. Multiple values for a rhs vector entry always arise for overlap equations; these are also added together. The second purpose of post-collocate is to make the computed matrix available in a form suitable for standard linear system solvers, by providing a compressed sparse row (CSR) format file ascii-coord, and a coordinate format file ascii-c-inp-csr.

The final result of all the work done by collocate and post-collocate (see Figure 6.1) is the matrix $[L]$ in CSR format.

6.2.5 Modules ascii-c-inp-csr, bin-full-data and ascii-coord

The files ascii-c-inp-csr, bin-full-data and ascii-coord contain the double precision matrix entries of $[L]$ in compressed sparse row, binary, and coordinate format, respectively. A typical block structure for a four-rectangle problem is shown in Figure 6.12 without block labels, and the first rectangle is magnified and shown with legible block labels in Figure 6.13.

These matrices have the block structure described in Section 4.5.2.

6.3 Solution

sp-solve is a simple wrapper around the SuperLU package[7] of (Demmel, Eisenstat, Gilbert, Li, and Liu 1999); it reads the data for $[L]$ from ascii-c-inp-csr, the data for $[f]$ from rhs-ascii-c-inp, and writes $[u]$ to solution-ascii as a single vector in double-precision format.

Although the original matrix $[L]$ is sparse with $O(m^3)$ entries[8], the direct factorization causes severe fill, reverting to $O(m^4)$ entries. Along with this fill, the work effort reverts to $O(m^6)$. See Section 8.1.3 for possible ways of avoiding these problems.

6.4 Reconstruction and evaluation

Just as the nested tree form for specification of input equations was necessary for practical use, so is the disassembly of the answer vector into its constituent unknown coefficients, and the encapsulation of those into simple, user-callable functions.

---

[7] At the time of this writing, version 2.0 is most recent, available at `http://www.nersc.gov/~xiaoye/SuperLU/superlu_2.0.tar.gz`.

[8] In practice, m is between 25 and 80

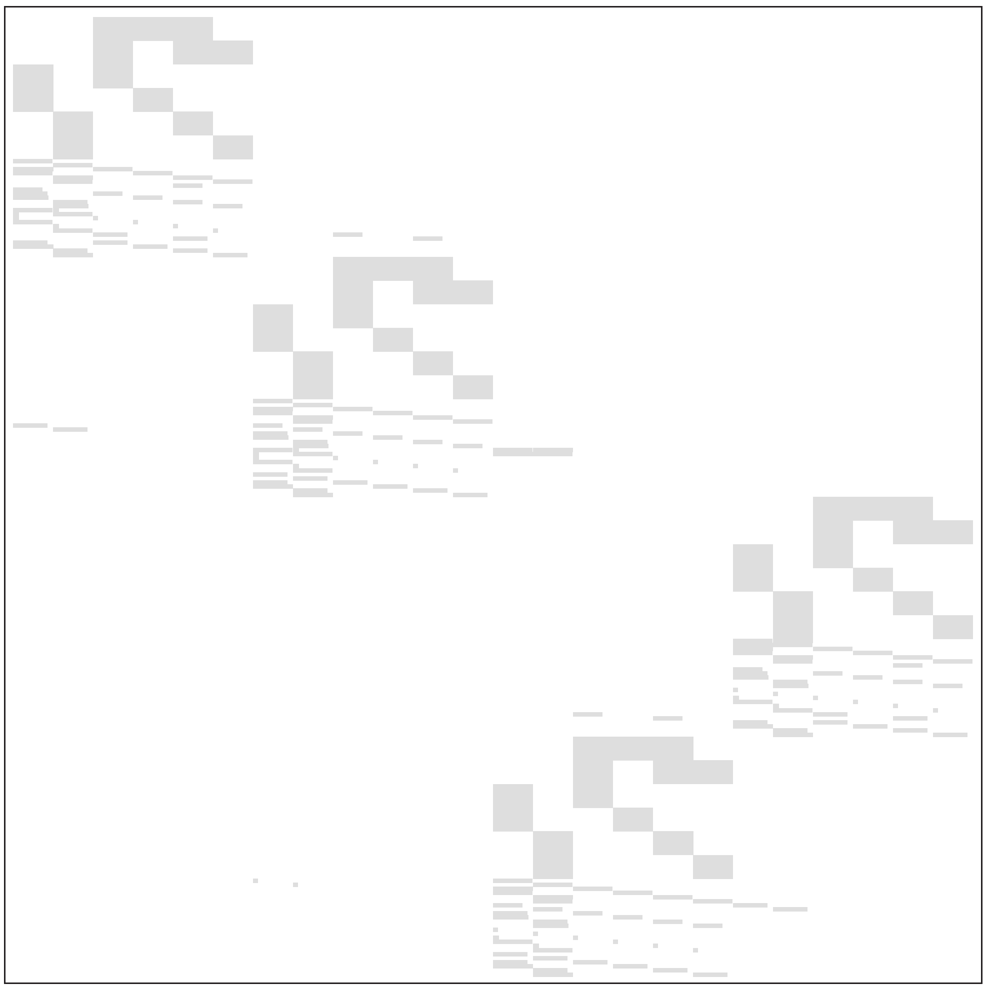


Figure 6.12. Matrix structure sample without labels. This picture shows one possibility for the contents of Figure 6.1, node ascii-coord. For a subset with labeled blocks, see Figure 6.13. All blocks are sparse themselves. $M = N = 3$, matrix size 1176×1176, 11884 nonzeros, 0.86% fill.

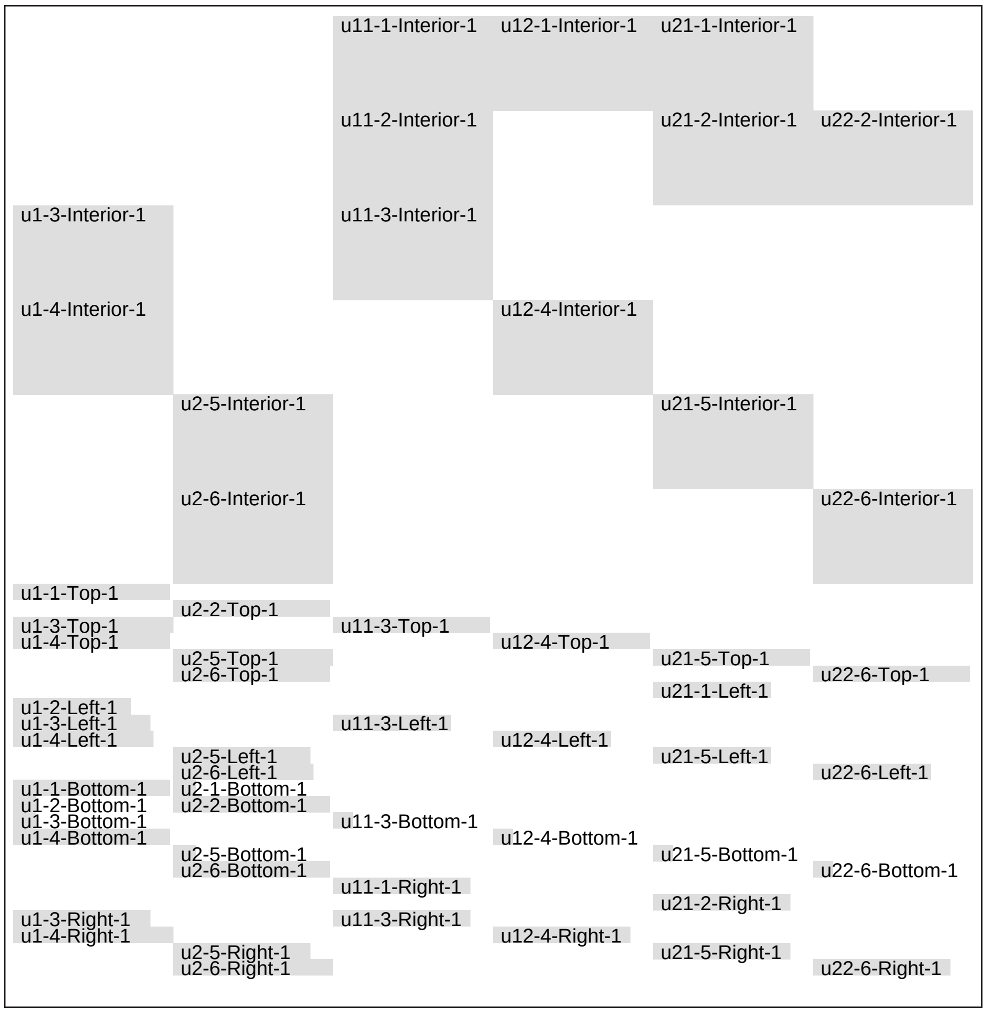


Figure 6.13. Matrix structure sample with labeled blocks for the domain 1 part of Figure 6.1, node ascii-coord. The full structure is shown in Figure 6.12. The label format is unknown-equation-region-domain. All blocks are sparse themselves. $M = N = 3$.

The user-provided files function-names and domain-geometry contain the list of functions to evaluate and the points at which to evaluate them. The eval_writeData program is first formed by ocamlc and incorporates the full geometry and sinc data from geometry.ml, other user data from global_data.ml, and the MAPLE parser from maple_parser_lib.cma. On execution, eval_writeData reads the full solution vector $[u]$ (from solution-ascii), the offsets of the c_{ij} in that vector (from bin-col-offset, and the output specifications (from function-names and domain-geometry); a regular grid is produced and written (in portable binary form) to data-grids.

6.4.1 Modules domain-geometry and function-names

The files function-names and domain-geometry specify which functions to form and where to evaluate them, respectively. Samples of these files are in Figure 6.14 and Figure 6.15. A typical grid on the rectangle $(0,1) \times (0,1)$ was shown in Figure 2.1 on page 16.

6.4.2 Module calc_value_funcs.cmo

The purpose of this module is the production of directly-callable interpolating functions $u_i(x,y)$. It forms these using the provided domain-specific sinc data, subvectors of the solution vector $[u]$, and auxiliary sinc-related functions.

6.4.3 Module eval_writeData

This executable reads the interpolants' coefficients c_{ij} from `solution-ascii`, uses the information in `bin-col-offset` to form the interpolants

$$u^{k|l} = \sum_i \sum_j c_{ij} \gamma_i(x) \gamma_j(y) \tag{6.5}$$

requested in `function-names`, and evaluates them at the points specified in `domain-geometry`.

The calculated data sets are stored in a collection of files, collectively called `data-grids` here.

6.4.4 Module data-grids

This is the collection of data files produced as the result of one run. Each file is in portable binary form and contains a value of the structure

```
type gridData =
  { position_x: float array;
    position_y: float array;
    yx_values: float array array }
```

```
full_info=[
   1=[
      x1=[
         num_pts=28,
         pts=[
            1.2328627002591529e-02, 2.0348051646723202e-02,
            3.9674829386731218e-02, 1.0692431112128870e-01,
            2.0519323334096654e-01, 3.0346215556064438e-01,
            4.0173107778032219e-01, 5.0000000000000000e-01,
            5.9826892221967787e-01, 6.9653784443935574e-01,
            7.9480676665903371e-01, 8.2789462830000005e-01,
            8.5782812710000000e-01, 8.9307568887871158e-01,
            9.1586246159999996e-01, 9.2616915422900004e-01,
            9.3785441989999996e-01, 9.4915422885599998e-01,
            9.6032517061327216e-01, 9.7023146959999995e-01,
            9.7965194835328495e-01, 9.8767137299739172e-01,
            9.8836515300000005e-01, 9.9099999999999999e-01,
            9.9174177687504261e-01, 9.9343283582099995e-01,
            9.9600000000000000e-01, 9.9800000000000000e-01],
         sinc_bounds=[1.4813528826400526e-02, 9.8518647117359948e-01],
         geometry_bounds=[0.0000000000000000e+00, 1.0000000000000000e+00]],
      x2=[
         num_pts=38,
         pts=[
            8.9478693000000003e-04, 1.0000000000000000e-03,
            ...
            1.0994605135293519e+01, 1.1829736756862346e+01],
         sinc_bounds=[3.9996527831281420e-01, 2.6600034721687184e+01],
         geometry_bounds=[0.0000000000000000e+00, 2.7000000000000000e+01]]],
      ... ]
```

Figure 6.14. Sample plot point specification, truncated for illustration. This input file corresponds to Figure 6.1, node domain-geometry, where it is used by eval_writeData. It is usually generated from other sources.

```
functions_to_plot=[
   [domain, name, operator]=[3, u1, I], [domain, name, operator]=[2, u1, I],
   [domain, name, operator]=[1, u1, I], [domain, name, operator]=[4, u2, D2],
   [domain, name, operator]=[3, u2, D2], [domain, name, operator]=[2, u2, D2],
   [domain, name, operator]=[1, u2, D2], [domain, name, operator]=[4, u2, D1],
   [domain, name, operator]=[3, u2, D1], [domain, name, operator]=[2, u2, D1],
   [domain, name, operator]=[1, u2, D1], [domain, name, operator]=[4, u2, I],
   [domain, name, operator]=[3, u2, I], [domain, name, operator]=[2, u2, I],
   [domain, name, operator]=[1, u2, I], [domain, name, operator]=[4, u1, D2],
   [domain, name, operator]=[3, u1, D2], [domain, name, operator]=[2, u1, D2],
   [domain, name, operator]=[1, u1, D2], [domain, name, operator]=[4, u1, D1],
   [domain, name, operator]=[3, u1, D1], [domain, name, operator]=[2, u1, D1],
   [domain, name, operator]=[1, u1, D1], [domain, name, operator]=[4, u1, I]]
```

Figure 6.15. Sample function name specification. This input file corresponds to Figure 6.1, node function-names, where it is used by eval_writeData.

CHAPTER 7

EXAMINATION OF THE QUESTION OF CONVERGENCE

This chapter presents definitions and theorems relevant to the algorithm and its convergence. There are three parts, preliminaries, solution results, and evaluation results.

The preliminaries unify and generalize existing results from several sources. Full proofs are provided for completeness.

The solution results comprise new definitions, requirements, and theorems relevant for the setup and solution of a linear algebraic system from a given linear elliptic system of PDEs. Definition 7.23 describes the structure of solvable systems; Theorem 7.24 and Theorem 7.25 state the requirements of solutions and coefficients, and give error bounds on the computed solution.

Lastly, new theorems concerning the use of computed results in the evaluation of unknowns at nonsinc points are provided; in particular, evaluation of functions of the proper class is no problem, but Theorem 7.27 also provides the justification for approximation of unbounded derivatives of these functions on a subset of the original domain.

The sinc interpolant is naturally defined on a region surrounding the real line; here, it is needed on a finite interval. These regions are more precisely defined in the following.

Definition 7.1 ($\mathcal{D}_d, \mathcal{D}, \phi, \psi$) *Pick $d > 0$ and define the strip $\mathcal{D}_d$ by*

$$\mathcal{D}_d = \{w \in \mathbb{C} : |\Im w| < d\} \tag{7.1}$$

Given a region $\mathcal{D}$ containing a contour Γ in $\mathbb{C}$ with endpoints a and b on the boundary of $\mathcal{D}$, ϕ is a one-to-one conformal map with the properties $\phi : \Gamma \to R$, $\phi(x) \to -\infty$ as $x \to a$, $\phi(x) \to \infty$ as $x \to b$ and $\phi : \mathcal{D} \to \mathcal{D}_d$

Define $\psi \equiv \phi^{-1}$; then the region $\mathcal{D}$ is the image

$$\mathcal{D} = \psi(\mathcal{D}_d) \tag{7.2}$$

Figure 7.1 shows a typical example for the functions $\phi(x) = \ln(x-a)/(b-x)$ and $\psi(z) = (\exp(z)b + a)/(\exp(z)+1)$ using values $d = \pi/3$, $a = 1$, and $b = 3$. These functions are also the ones used in the remainder of this work; the eye-shaped region shown in the figure will be referred to as $\mathcal{D}_e$. Many other maps are available; see (Stenger 1993a), Section 1.7.

The sinc interpolant naturally interpolates exponentially decaying functions on the real line. The following definition generalizes that interpolation ability to a class of functions, via a use of the previously defined conformal map, ϕ.

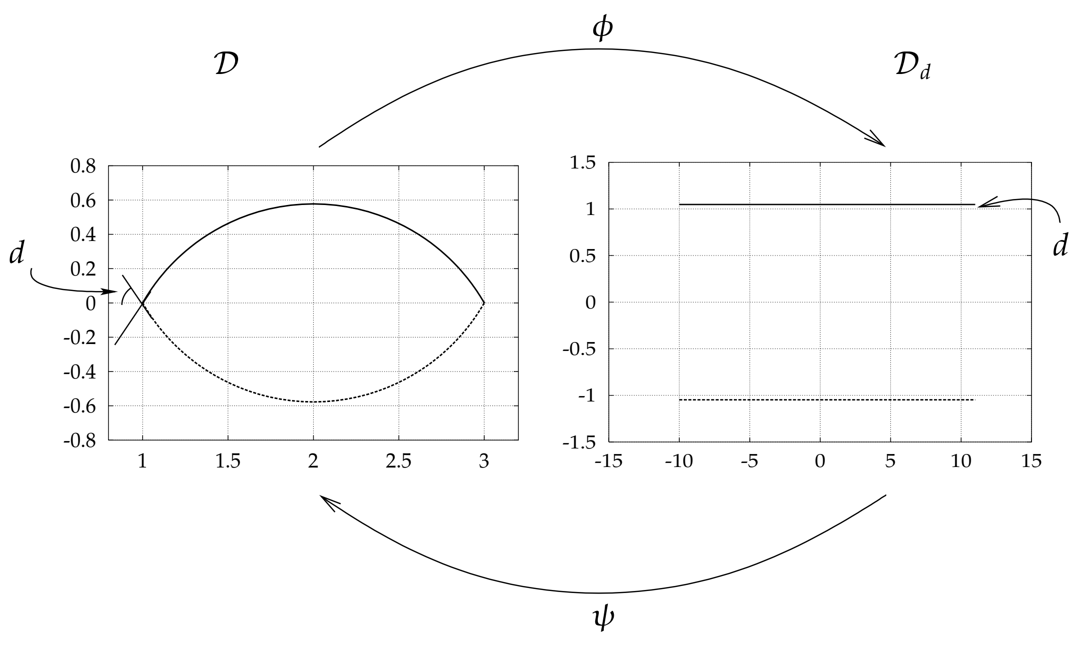


Figure 7.1. A part of the strip-shaped infinite region $\mathcal{D}_d$ and its image $\mathcal{D}$, for $\phi(x) = \ln(x-a)/(b-x)$, $\psi(z) = (\exp(z)b + a)/(\exp(z)+1)$, $d = \pi/3$, $a = 1$, $b = 3$. The points $kh : k \in Z$ are mapped to $[a, b]$.

Definition 7.2 (ρ, L_α) *For the region $S_d = \{z \in \mathbb{C} : |\arg z| < d\}$, the map $\rho : \mathcal{D} \to S_d$ is defined as*

$$\rho(z) \; = e^{\phi(z)} \tag{7.3}$$

Notice that for $z \in \mathbb{R}$, $\rho : \Gamma \to [0, \infty)$. Given $\alpha > 0$, $d > 0$ and a region $\mathcal{D}$, denote by L_α the family of all functions $F(z)$ analytic and uniformly bounded in $\mathcal{D}$ so that $\forall z \in \mathcal{D}$,

$$|F(z)| \; \le \frac{C|\rho(z)|^\alpha}{|1+\rho(z)|^{2\alpha}} \tag{7.4}$$

for some $C > 0$.

On $\mathbb{R}$ using $\phi(z) = z$, this criterion is

$$|F(z)| \; \le \frac{C|e^z|^\alpha}{|1+e^z|^{2\alpha}} \tag{7.5}$$

so as $z \to \infty$, $|F(z)| \le C_1|e^{-\alpha z}|$ and as $z \to -\infty$, $|F(z)| \le C_2|e^{\alpha z}|$ and L_α is the class of exponentially decaying functions.

On $[a, b]$ using

$$\phi(z) \; = \ln(z-a)/(b-z), \tag{7.6}$$

$\psi(z)$ is given by

$$\psi(z) \; = \frac{e^z b + a}{e^z + 1} \tag{7.7}$$

and

$$|F(z)| \; \le C\left|\frac{z-a}{b-z}\right|^\alpha \left|\frac{b-z}{b-a}\right|^{2\alpha} = C_1|z-a|^\alpha |b-z|^\alpha \tag{7.8}$$

so as $z \to a^+$, $|F(z)| \le C_2|z-a|^\alpha$ and as $z \to b^-$, $|F(z)| \le C_3|b-z|^\alpha$. Thus, algebraic decay near the endpoints is required of L_α functions in this case.

The following space is useful in several theorems, but is not otherwise used.

Definition 7.3 (Spaces) $\mathbf{H}^\infty(\mathcal{D})$ *is the space of all functions f analytic and uniformly bounded in $\mathcal{D}$, i.e.,*

$$\sup_{z\,\in\mathcal{D}} |f(z)| \; < \infty. \tag{7.9}$$

Define $N(f, \mathcal{D})$ as

$$N(f, \mathcal{D}) \; \equiv \int_{\mathcal{D}} |f(z)||dz|. \tag{7.10}$$

Then $\mathbf{H}^1(\mathcal{D})$ is the space of all functions f such that f is analytic in $\mathcal{D}$ and $N(f, \mathcal{D}) < \infty$.

As an extension of the preceding definitions, the general class of functions dealt with by the sinc interpolant can now be defined.

Definition 7.4 (M_α) *Let $\alpha \in (0,1]$ and $d \in (0,\pi)$. Define f as*

$$f(z) \ = g(z) - t_L(x)g(a) - t_R(x)g(b) \tag{7.11}$$

Then $M_\alpha(\mathcal{D})$ denotes the family of functions $g \in \mathbf{H}^\infty(\mathcal{D})$ such that $f \in L_\alpha(\mathcal{D})$.

On $[a,b]$, this definition reduces to

$$f \ = g - \left[\frac{(b-z)g(a) + (z-a)g(b)}{b-a}\right] \tag{7.12}$$

Functions in the $M_\alpha(\mathcal{D})$ class may have nonzero values at the endpoints, and this class is used in the remainder of this work.

The following definition presents a concise summary of one-dimensional approximation. Omitted for now are the precise conditions on the function to be approximated, and the expected convergence rate. Those are the subject of the later proofs.

Definition 7.5 (Approximation in one dimension) *For a given $N > 0$, the j^{th} sinc point is given by*

$$z_j \ = \phi^{-1}(jh) = \psi(jh) \tag{7.13}$$

where the sinc spacing parameter is

$$h \ = \sqrt{\frac{\pi d}{\alpha N}}. \tag{7.14}$$

The sinc function is defined as

$$\mathrm{sinc}(x) \ = \frac{\sin(\pi x)}{\pi x}; \tag{7.15}$$

using the definitions

$$t_L(x) \ = \frac{1}{1+e^x} \circ \phi(x), \tag{7.16}$$

$$S(j,h)(x) \ = \mathrm{sinc}\left(\frac{x-jh}{h}\right), \tag{7.17}$$

$$S_j(x) = S(j,h) \circ \phi(x), \tag{7.18}$$

$$t_R(x) = \frac{e^x}{1+e^x} \circ \phi(x), \tag{7.19}$$

and

$$\omega_k(x) = \begin{cases} t_L(x) - \sum_{j=-N+1}^{N} t_L(x_j) S_j(x) & k = -N \\ S_k(x) & k \in -N+1..N-1\,, \\ t_R(x) - \sum_{j=-N}^{N-1} t_R(x_j) S_j(x) & k = N \end{cases} \tag{7.20}$$

the sinc interpolant on Γ *is defined by*

$$\sum_{k=-N}^{N} f(z_k)\omega_k(x). \tag{7.21}$$

The names $t_L(x)$ and $t_R(x)$ indicate the use of these spline-like terms to represent the interpolant's value at the left, respectively right, endpoint of an interval.

In the following, the norm $\|\cdot\|$ is the maximum norm on Γ, i.e., for $f(z) \in \Gamma$,

$$\|f(z)\| = \max_{z\in\Gamma} |f(z)|, \tag{7.22}$$

and $C, C_1, C_2, \ldots$ denote positive constants independent of the number of terms in the associated series (N, $N^{|i,x}$ and $N^{|i,x}$). Constants of the same name occurring in different theorems are, of course, distinct.

For brevity, define the operators

$$(\overline{T}f)(x) = f(a)t_L(x) + f(b)t_R(x), \tag{7.23}$$

$$(Tf)(x) = f(x_{-N})t_L(x) + f(x_N)t_R(x) \tag{7.24}$$

and their derivatives

$$(\overline{T}'f)(x) = f(a)t'_L(x) + f(b)t'_R(x) \tag{7.25}$$

and

$$(T'f)(x) = f(x_{-N})t'_L(x) + f(x_N)t'_R(x). \tag{7.26}$$

For $x \in \Gamma$, let $y = \phi(x) \in \mathbb{R}$ and note the following bounds:

$$
\begin{aligned}
|t_L(x)| &= \left|\frac{1}{e^y + 1}\right| \leq 1 \\
|t_R(x)| &= \left|\frac{1}{e^{-y} + 1}\right| \leq 1 \\
\left|\frac{t_L'(x)}{\phi'(x)}\right| &\leq \frac{1}{4} \\
\left|\frac{t_R'(x)}{\phi'(x)}\right| &\leq \frac{1}{4}
\end{aligned}
\tag{7.27}
$$

The following lemma simply shows that the left "spline" has near-zero value (for large N) at the rightmost sinc point, and vice versus. This is then used in the next lemma.

Lemma 7.6 (Orthogonalization residual) *For $N > 1$,*

$$|t_L(x_N)| \leq e^{-\sqrt{\pi d \alpha N}} \tag{7.28}$$

$$|t_R(x_{-N})| \leq e^{-\sqrt{\pi d \alpha N}} \tag{7.29}$$

Proof: For the first part,

$$|t_L(x_N)| = \frac{1}{e^{Nh} + 1} \leq \frac{1}{e^{Nh}} \leq e^{-\sqrt{\pi d \alpha N}} \tag{7.30}$$

as $\alpha \in (0, 1]$. A similar inequality holds for the second part. • • •

The following lemma is the connection between prior work by (Stenger 1993a), Section 6.5, and (Schwing 1976), which used function values at endpoints, and the present work. In the former, special sinc points on the boundary of a domain were used, while in the present work, all points are regular interior sinc points. This is also the reason for the closure-like notation $\overline{T}g$ for boundary-including approximations, and Tg for strictly interior approximations.

Lemma 7.7 (Projection distance) *Given a function $g \in M_\alpha(\mathcal{D})$,*

$$|(\overline{T}g)(x) - (Tg)(x)| \leq Ce^{-\sqrt{\pi d \alpha N}} \tag{7.31}$$

and

$$\left|\frac{1}{\phi'(x)}\left[(\overline{T}'g)(x) - (T'g)(x)\right]\right| \leq Ce^{-\sqrt{\pi d \alpha N}} \tag{7.32}$$

Proof: Notice that

$$|(\overline{T}g)(x) - (Tg)(x)| = |[g(a) - g(x_{-N})]t_L(x) + [g(b) - g(x_N)]t_R(x)| \tag{7.33}$$

and

$$\begin{aligned} \Big|\frac{1}{\phi'(x)}\Big[(\overline{T}'g)(x) - (T'g)(x)\Big]\Big| &= \\ &\Big|\frac{1}{\phi'(x)}\Big|\,\Big|[g(a) - g(x_{-N})]t'_L(x) + [g(b) - g(x_N)]t'_R(x)\Big|, \end{aligned} \tag{7.34}$$

so Equations 7.31 and 7.32 will follow if $|g(a) - g(x_{-N})|$ and $|g(b) - g(x_N)|$ can be bounded, as the remaining terms are already bounded in Equation 7.27. To show $|g(a) - g(x_{-N})| \le Ce^{-\sqrt{\pi d\alpha N}}$, use $g \in M_\alpha$ to obtain

$$|g(z_{-N}) - t_L(z_{-N})g(a) - t_R(z_{-N})g(b)| \le Ce^{-Nh\alpha} \le Ce^{-\sqrt{\pi d\alpha N}}, \tag{7.35}$$

observe that

$$|t_L(z_{-N}) - 1| = \Big|\frac{e^{-Nh}}{1 + e^{-Nh}}\Big| \le e^{-Nh} \le e^{-\sqrt{\pi d\alpha N}}, \tag{7.36}$$

and obtain the bound

$$\begin{aligned} |g(a) - g(z_{-N})| &= \\ &|g(z_{-N}) - t_L(z_{-N})g(a) - t_R(z_{-N})g(b) \\ &\quad + t_R(z_{-N})g(b) + (t_L(z_{-N}) - 1)g(a)| \\ &\le e^{-\sqrt{\pi d\alpha N}}(C + g(b) + g(a)) \end{aligned} \tag{7.37}$$

using Equation 7.29. The bound for $|g(b) - g(x_N)|$ is derived analogously. • • •

In the next lemma, $\Psi(n)$ is the digamma function, defined by

$$\Psi(z) = \frac{d}{dz}\Gamma(z) = \Gamma'(z)/\Gamma(z). \tag{7.38}$$

For integral k and n,

$$\Psi(n+1) = \sum_{k=1}^{n} \frac{1}{k} - \gamma, \tag{7.39}$$

but sums of the form $\sum_{k=-N}^{N} \frac{1}{\pi(k-x)}$, $x \in \mathbb{R}$, are needed. To this end, the recurrence relation

$$\Psi(x+1) = \Psi(x) + \frac{1}{x}, \tag{7.40}$$

valid for $x \in \mathbb{R}$, can be used to obtain the formula

$$\sum_{p=n}^{k+n} \frac{1}{x-p} = \Psi(x+1-n) - \Psi(x-k-n) \tag{7.41}$$

for $x > p$ and $k, n, p \in \mathbb{Z}$ with $k > 0$, and its twin

$$\sum_{p=n-k}^{n} \frac{1}{p-x} = \Psi(1+n-x) - \Psi(n-k-x). \tag{7.42}$$

for $x < p$.

This new lemma is used repeatedly to prove convergence rates for interpolation and differentiation; first, for generalized one-dimensional results, and later for two-dimensional theorems.

Lemma 7.8 (Finite sum bound) *For $x \in \mathbb{R}$,*

$$\begin{aligned}\sum_{k=-N}^{N} |S(k,h)(x)| &\leq 2\,\frac{\Psi(N+3)}{\pi} + 2\,\frac{\gamma}{\pi} + 2 - 1/\pi \\ &\leq C_1 \ln(N) + C_2\end{aligned} \tag{7.43}$$

and

$$\begin{aligned}\sum_{k=-N}^{N} |hS'(k,h)(x)| &\leq [1 + \frac{1}{2\pi}]\left(2\,\frac{\Psi(N+3)}{\pi} + 2\,\frac{\gamma}{\pi} + 2 - 1/\pi\right) \\ &\leq C_3 \ln(N) + C_4\end{aligned} \tag{7.44}$$

Proof: First bound the sum of values. For this, note the following piecewise bounds for $\mathrm{sinc}(x)$:

$$|\mathrm{sinc}(x)| \leq \begin{cases} 1 & \text{for } |x| < 1 \\ |\frac{1}{x}| & \text{for } |x| \geq 1 \end{cases} \tag{7.45}$$

Notice the sum in Equation 7.43 is even. This follows directly from the observation $|S(k,h)(x)| = |S(-k,h)(-x)|$ and some series rearrangement.

Next, consider two regions,

$$R_o = \{x \,|\, -\infty < xh \leq -N-1\} \tag{7.46}$$

and

$$R_i = \{x \,|\, -N-1 < xh \leq 0\} \tag{7.47}$$

Define B_o and B_i by

$$B_o = \frac{\Psi(N+1-x)}{\pi} - \frac{\Psi(-N-x)}{\pi} \tag{7.48}$$

and

$$B_i = \frac{\Psi(x+N+3)}{\pi} + 2\,\frac{\gamma}{\pi} + 2 - 1/\pi + \frac{\Psi(N+3-x)}{\pi}, \tag{7.49}$$

respectively. By definition, $\Psi'(x) = \psi_1(x)$, and thus

$$B_o' = -\frac{\psi_1(N+1-x)}{\pi} + \frac{\psi_1(-N-x)}{\pi} \tag{7.50}$$

and

$$B_i' = \frac{\psi_1(x+N+3)}{\pi} - \frac{\psi_1(N+3-x)}{\pi} \tag{7.51}$$

Then on R_o,

$$\sum_{k=-N}^{N} |S(k,h)(xh)| \le \sum_{k=-N}^{N} \frac{1}{\pi\ (k-x)} = B_o(x) \tag{7.52}$$

Now, $\psi_1(x)$ is monotone decreasing for $x > 0$, so $B_o' > 0$ and therefore $B_o(x)$ increases monotonically, and must have its largest value at the right boundary. On the right boundary,

$$B_o(-N-1) = \frac{\Psi(2\,N+2)}{\pi} + \frac{\gamma}{\pi} \tag{7.53}$$

On R_i,

$$\begin{aligned}
\sum_{k=-N}^{N} |S(k,h)(xh)| &\le \sum_{k=-N-2}^{N+2} |S(k,h)(xh)| \\
&\le \sum_{k=-N-2}^{\lfloor x\rfloor -1} \frac{1}{\pi(x-k)} + 2 + \sum_{k=\lfloor x\rfloor+2}^{N+2} \frac{1}{\pi(k-x)} \\
&= \frac{\Psi(x+N+3)}{\pi} + 2 + \frac{\Psi(N+3-x)}{\pi} \\
&\quad - \frac{\Psi(1+x-\lfloor x\rfloor) + \Psi(2-(x-\lfloor x\rfloor))}{\pi} \\
&\le B_i(x);
\end{aligned} \tag{7.54}$$

the last inequality follows from using the property $0 \le x - \lfloor x\rfloor \le 1$ to find the minimum of $[\Psi(1+x-\lfloor x\rfloor) + \Psi(2-(x-\lfloor x\rfloor))]$ to be $1-2\gamma$. By the same monotonicity argument as used above, B_i has its maximum value on the right boundary, and

$$B_i(0) = 2\,\frac{\Psi(N+3)}{\pi} + 2\,\frac{\gamma}{\pi} + 2 - 1/\pi \tag{7.55}$$

Because $\Psi(x) \approx \ln(x)$ for large x, $B_i(0) > B_o(-N-1)$, giving the bound in Equation 7.43.

Now, bound the derivative sum. For $|x| \geq 2$, $\mathrm{sinc}'(x)$ can be bounded as follows.

$$\begin{aligned} |\mathrm{sinc}'(x)| &= \left|\frac{\cos(\pi\, x)}{x} - \frac{\sin(\pi\, x)}{\pi\, x^2}\right| \\ &= \left|\frac{1}{x}\right| \left|\cos(\pi x) - \frac{\sin(\pi x)}{\pi x}\right| \\ &\leq \left|\frac{1}{x}\right| \left|1 + \frac{1}{2\pi}\right| \end{aligned} \tag{7.56}$$

For $0 < |x| < 2$, the same bound is easily verified empirically. As

$$hS'(k,h) = \mathrm{sinc}'(x/h - k), \tag{7.57}$$

Equation 7.44 now follows from the preceding argument using $\left[1 + \frac{1}{2\pi}\right]$ as the scale in Equation 7.45. • • •

The following two theorems show the sinc interpolant to have exponential convergence when interpolating functions in the right class. Both theorems are stated in (Kowalski, Sikorski, and Stenger 1995), but only a partial proof is given there. As they are a critical part of the present work, full proofs are provided here.

Theorem 7.9 (Interpolation) *If $f \in M_\alpha(\mathcal{D})$, then*

$$\|f - \sum_{j=-N}^{N} f(z_j)\omega_j\| \leq C\sqrt{N}e^{-\sqrt{\pi d \alpha N}} \tag{7.58}$$

Proof: This is shown in two parts. First, a bound is obtained for $g(x) \in L_\alpha$; using this bound, Equation 7.58 is derived.

Part A: Theorem 119.1 of (Kowalski, Sikorski, and Stenger 1995) shows that $g(z) \in L_\alpha$ implies $\phi'(z)g(z) \in \mathbf{H}^1(\mathcal{D})$. Using Theorem 126.1 of (Kowalski, Sikorski, and Stenger 1995), with $F(z) = \phi'(z)g(z)$, the equation

$$\begin{aligned} g(x) - \sum_{k\in\mathbb{Z}} g(x_k)S_k(x) \\ = \frac{1}{2\pi i}\int_{\partial\mathcal{D}} \phi'(z)g(z)\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}\frac{dz}{\sin(\pi\phi(z)/h)} \end{aligned} \tag{7.59}$$

follows.

To bound the integral, consider the following parts. For $z \in \partial\mathcal{D}$, $\phi(z) = \pm id$ and

$$\left|\frac{1}{\sin(\pi\phi(z)/h)}\right| \le \left|\frac{1}{\sinh(\pi d/h)}\right| \le Ce^{-\sqrt{\pi d\alpha N}} \tag{7.60}$$

By definition, $\phi'(z)g(z) \in \mathbf{H}^1(\mathcal{D})$ gives

$$\int_{\partial\mathcal{D}} |\phi'(z)g(z)|\,|dz| \le C \tag{7.61}$$

For $z \in \partial\mathcal{D}$, $\Im\phi(z) = \pm d$ and for $x \in \Gamma$, $\phi(x) \in \mathbb{R}$, so

$$|\phi(z) - \phi(x)| \ge d \tag{7.62}$$

and thus

$$\left|\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}\right| \le 1/d \tag{7.63}$$

To bound the tails of the summation, start with the identity

$$S(k,h)(x) = \frac{h}{2\pi}\int_{-\pi/h}^{\pi/h} e^{(-it(kh-x))}dt \tag{7.64}$$

and obtain the bound for $|S_k(x)|$

$$|S_k(x)| = \left|\frac{1}{2}\int_{-1}^{1} e^{-iu\pi(kh-\phi(x))/h}du\right| \le 1 \tag{7.65}$$

Further, $g \in L_\alpha$ implies

$$|g(x_k)| \le Ce^{-h\alpha|k|}, \tag{7.66}$$

which, together with the identities

$$\sum_{k=1}^{N} e^{kx} = e^x \frac{e^{nx}-1}{e^x-1} \tag{7.67}$$

and

$$\sum_{k=0}^{\infty} e^{-kx} = \frac{1}{1-e^{-x}}, \tag{7.68}$$

results in

$$\sum_{|k|>N} |g(x_k)S_k(x)| \le Ce^{-h\alpha N}\frac{e^{-h\alpha}}{1-e^{-h\alpha}} \le C\sqrt{N}e^{-\sqrt{\pi d\alpha N}} \tag{7.69}$$

The last inequality is obtained from a power series expansion of $e^{h\alpha}$.

Combining these results,

$$\begin{aligned}
&\left|g(x) - \sum_{k=-N}^{N} g(x_k)S_k(x)\right| \\
&\quad = \left|\sum_{|k|>N} g(x_k)S_k(x) + \frac{1}{2\pi i}\int_{\partial\mathcal{D}} \frac{\phi'(z)g(z)}{\sin(\pi\phi(z)/h)}\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}dz\right| \\
&\quad \le C\sqrt{N}e^{-\sqrt{\pi d\alpha N}}
\end{aligned} \tag{7.70}$$

Part B: For any $x \in [a,b]$ and $f \in M_\alpha$,

$$\begin{aligned}
&|f(x) - \sum_{k=-N}^{N} f(x_k)\omega_k(x)| \\
&\quad = |f(x) - (Tf)(x) - \sum_{k=-N+1}^{N-1} [f(x_k) - (Tf)(x_k)]S_k(x) \\
&\quad\quad + f(x_{-N})t_L(x_N)S_N(x) + f(x_N)t_R(x_{-N})S_{-N}(x)| \\
&\quad = |f(x) - (\overline{T}f)(x) - \sum_{k=-N+1}^{N-1} [f(x_k) - (\overline{T}f)(x_k)]S_k(x) \\
&\quad\quad + f(x_{-N})t_L(x_N)S_N(x) + f(x_N)t_R(x_{-N})S_{-N}(x) \\
&\quad\quad - (Tf)(x) + (\overline{T}f)(x) \\
&\quad\quad + \sum_{k=-N+1}^{N-1} [(\overline{T}f)(x_k) - (Tf)(x_k)]S_k(x)| \\
&\quad \le e^{-\sqrt{\pi d\alpha N}}\{C_1\sqrt{N} + C_2 + C_3 + C_4 + (C_5\sqrt{N} + C_6)\} \\
&\quad \le C\sqrt{N}e^{-\sqrt{\pi d\alpha N}}
\end{aligned} \tag{7.71}$$

The C_1 term comes from Equation 7.70 because $g(x) \equiv f(x) - (\overline{T}f)(x) \in L_\alpha$, C_2 and C_3 stem from Lemma 7.6, C_4 originates from Lemma 7.7, and C_5 and C_6 come from Lemma 7.8. • • •

Theorem 7.10 (Differentiation) *Let $f \in M_\alpha(\mathcal{D}')$. Then*

$$\left\| \left(\frac{h}{\phi'}\right) \left[f - \sum_{j=-N}^{N} f(z_j)\omega_j \right]' \right\| \leq C\sqrt{N}e^{-\sqrt{\pi d \alpha N}} \tag{7.72}$$

Proof: This proof follows that of Theorem 7.9, with appropriate modifications made for derivatives; again, start with a bound for $g \in L_\alpha$.

Part A: Differentiating and scaling both sides of Equation 7.59 gives

$$\begin{aligned} &\frac{h}{\phi'(x)}[g'(x) - \sum_{k\in\mathbb{Z}} g(x_k)S_k'(x)] \\ &\qquad = \frac{h}{\phi'(x)}\frac{1}{2\pi i}\int_{\partial\mathcal{D}} \phi'(z)g(z)\left[\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}\right]' \frac{dz}{\sin(\pi\phi(z)/h)} \end{aligned} \tag{7.73}$$

Inequalities 7.60 and 7.61 remain unchanged, so to bound the integral, only the bound

$$\left| [\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}]' \right| \leq \left|\frac{\phi'(x)}{h}\right| (\frac{\pi}{d} + 1/d^2) \tag{7.74}$$

is needed. This bound follows from direct differentiation and Equation 7.62; obviously,

$$\left| \frac{h}{\phi'(x)}[\frac{\sin(\pi\phi(x)/h)}{\phi(z)-\phi(x)}]' \right| \leq C. \tag{7.75}$$

To bound the tails of the summation, differentiate the identity in Equation 7.64 and obtain

$$|S_k'(x)| = \left|\frac{\phi'(x)}{h}\right| \left| \frac{i\pi}{2}\int_{-1}^{1} ue^{-iu\pi(kh-\phi(x))/h}du \right|, \tag{7.76}$$

from which

$$\left|\frac{h}{\phi'(x)}\right| |S_k'(x)| \leq \pi/2 \tag{7.77}$$

immediately follows. Again, $g \in L_\alpha$ implies

$$|g(x_k)| \leq Ce^{-h\alpha|k|}, \tag{7.78}$$

which, together with identities 7.67 and 7.68, results in

$$\sum_{|k|\ >N} \left| g(x_k) S_k'(x) \frac{h}{\phi'(x)} \right| \ \leq C\sqrt{N} e^{-\sqrt{\pi d \alpha N}} \tag{7.79}$$

Combining these bounds,

$$\left| \frac{h}{\phi'(x)} \right| \left| g'(x) - \sum_{k\ =-N}^{N} g(x_k) S_k'(x) \right| \ \leq C\sqrt{N} e^{-\sqrt{\pi d \alpha N}} \tag{7.80}$$

Part B: For any $x \in [a, b]$ and $f \in M_\alpha$,

$$\begin{aligned}
&\left| \frac{h}{\phi'(x)} \right| |f'(x) - \sum_{k=-N}^{N} f(x_k) \omega_k'(x)| \\
&\quad = \left| \frac{h}{\phi'(x)} \right| |f'(x) - (T'f)(x) \\
&\quad - \sum_{k=-N+1}^{N-1} [f(x_k) - (Tf)(x_k)] S_k'(x) \\
&\quad + f(x_{-N}) t_L(x_N) S_N'(x) + f(x_N) t_R(x_{-N}) S_{-N}'(x)| \\
&\quad = | \frac{h}{\phi'(x)} \Big\{ f'(x) - (\overline{T}' f)(x) \\
&\qquad - \sum_{k=-N+1}^{N-1} [f(x_k) - (\overline{T} f)(x_k)] S_k'(x) \Big\} \\
&\quad + \frac{h}{\phi'(x)} \Big\{ f(x_{-N}) t_L(x_N) S_N'(x) + f(x_N) t_R(x_{-N}) S_{-N}'(x) \\
&\qquad - (T'f)(x) + (\overline{T}' f)(x) \\
&\qquad + \sum_{k=-N+1}^{N-1} [(\overline{T} f)(x_k) - (Tf)(x_k)] S_k'(x) \Big\} | \\
&\quad \leq \ e^{-\sqrt{\pi d \alpha N}} \{ C_1 \sqrt{N} + C_2 + C_3 + C_4 + \{ (C_5 \sqrt{N} + C_6) \} \\
&\quad \leq C\sqrt{N} e^{-\sqrt{\pi d \alpha N}}
\end{aligned} \tag{7.81}$$

The C_1 term comes from Equation 7.80 because $g(x) \equiv f(x) - (\overline{T} f)(x) \in L_\alpha$, C_2 and C_3 stem from Lemma 7.6 and Equation 7.77, C_4 originates from Lemma 7.7, and C_5 and C_6 come from Lemma 7.8. • • •

The two preceding theorems are the last one-dimensional results needed here. The remainder of this work deals with multiple dimensions (two), unknowns, domains, and equations. The next three definitions provide a precise naming convention.

To uniquely label unknowns (and their series terms), domains, equations and regions (and their associated collocation points), introduce the following notation.

Definition 7.11 (Index notation) *Assuming unknowns and directions are indexed, the notation*

$$i|j,k \tag{7.82}$$

denotes unknown i in domain j and direction k. Usually, i or k will be absent. Similarly, the notation

$$i \wedge R \tag{7.83}$$

denotes equation i in region R

Using this notation, the two-dimensional collocation points, basis functions, operators and right-hand sides are fully described by the following definitions.

Since a given problem consists of main equations and boundary conditions, it is natural to partition its discrete version, specified by the collocation points, into analogous regions. This gives rise to the following definition.

Definition 7.12 (Collocation points and regions) *The collocation points are given by $z^{|j}$, the array of points for the j^{th} domain with entries*

$$z_{kl}^{|j} = (\psi^{|j,x}(kh^{|j,x}), \psi^{|j,y}(lh^{|j,y})) = (x_k^{|j}, y_l^{|j}) \tag{7.84}$$

These collocation points are partitioned into regions analogous to the equations and boundary conditions of the problem class. On every rectangular domain, we thus have the regions top (T*), right (*R*), left (*L*), bottom (*B*), and interior (*I*).*

To handle multiple domains, we additionally introduce the regions top-, right-, bottom- and left-overlap (T°*,* R°*,* B° *and* L°*, respectively). Their locations are identical to the regions* T*,* R*,* B *and* L*, respectively, but their meaning and use is different.*

Definition 7.13 (Unknown representations) *The i^{th} unknown in the j^{th} domain is denoted by*

$$u^{i|j}(x,y). \tag{7.85}$$

Using the preceding notation, the necessary two-dimensional extensions for sinc interpolation can now be defined. The idea of the $X_{2\alpha}$ product space is taken from (Schwing 1976), where it was used for integral equations.

Definition 7.14 (Two-dimensional spaces) *Rectangle i is denoted by $Q^{|i}$ and defined as*

$$Q^{|i} = [a^{|i,x}, b^{|i,x}] \times [a^{|i,y}, b^{|i,y}] \tag{7.86}$$

Make the definitions

$$\begin{aligned} \mathcal{D}^{|i,x} &= \psi^{|i,x}(\mathcal{D}_d^{|i,x}) && (7.87)\\ \mathcal{D}^{|i,y} &= \psi^{|i,y}(\mathcal{D}_d^{|i,y}) && (7.88)\\ \Omega^{|i} &= \left\{[a^{|i,x}, b^{|i,x}] \times \overline{\mathcal{D}^{|i,y}}\right\} \cup \left\{\overline{\mathcal{D}^{|i,x}} \times [a^{|i,y}, b^{|i,y}]\right\} && (7.89) \end{aligned}$$

and for any $x \in [a^{|i,x}, b^{|i,x}]$, and given a function $f^{|i} : \Omega^{|i} \to \mathbb{C}$ with $f^{|i} \in \mathbf{H}^\infty(Q^{|i})$, define

$$f_2^{|i}(x,y) : \overline{\mathcal{D}^{|i,y}} \to \mathbb{C} \tag{7.90}$$

by

$$f_2^{|i}(x,y) = f^{|i}(x,y). \tag{7.91}$$

Similarly, for any $y \in [a^{|i,y}, b^{|i,y}]$, define

$$f_1^{|i}(x,y) : \overline{\mathcal{D}^{|i,x}} \to \mathbb{C} \tag{7.92}$$

by

$$f_1^{|i}(x,y) = f^{|i}(x,y). \tag{7.93}$$

Define the space $X_{2\alpha}$ by

$$\begin{aligned} X_{2\alpha} = \{f^{|i} \text{ on } \Omega^{|i} \quad | \quad & f_1^{|i} \in M_\alpha(\overline{\mathcal{D}^{|i,x}}) \forall y \in [a^{|i,y}, b^{|i,y}] \text{ and} && (7.94)\\ & f_2^{|i} \in M_\alpha(\overline{\mathcal{D}^{|i,y}}) \forall x \in [a^{|i,x}, b^{|i,x}]\} && (7.95) \end{aligned}$$

To prove convergence for higher-dimensional approximations, a factorization of the interpolant is used, and that factorization's parts are then bounded.

To obtain this factorization, the following definition's projection operators are needed.

The basic idea is taken from (Schwing 1976), but is here extended to (a) use only interior points, (b) also approximate derivatives, and (c) work on any M_α product space, not just a finite rectangle.

Definition 7.15 (Projection operators) *For every $Q^{|i}$, and points $x, y \in Q^{|i}$, define the following operators for function approximation:*

$$(\Pi^{|i,1} f^{|i})(x) = \sum_{k=-N^{|i,x}}^{N^{|i,x}} f_1^{|i}(x_k^{|i}, y)\omega_k^{|i,x}(x) \tag{7.96}$$

$$(\Pi^{|i,2} f^{|i})(y) = \sum_{k=-N^{|i,y}}^{N^{|i,y}} f_2^{|i}(x, y_k^{|i})\omega_k^{|i,y}(y) \tag{7.97}$$

$$(\overline{T}_1^{|i} f^{|i})(x) = f_1^{|i}(a^{|i,x}, y)t_L^{|i,x}(x) + f_1^{|i}(b^{|i,x}, y)t_R^{|i,x}(x) \tag{7.98}$$

$$(T_1^{|i} f^{|i})(x) = f_1^{|i}(x_{-N^{|i,x}}^{|i}, y)t_L^{|i,x}(x) + f_1^{|i}(x_{N^{|i,x}}^{|i}, y)t_R^{|i,x}(x) \tag{7.99}$$

$$(\overline{T}_2^{|i} f^{|i})(y) = f_2^{|i}(x, a^{|i,y})t_L^{|i,y}(y) + f_2^{|i}(x, b^{|i,y})t_R^{|i,y}(y) \tag{7.100}$$

$$(T_2^{|i} f^{|i})(y) = f_2^{|i}(x, y_{-N^{|i,y}}^{|i})t_L^{|i,y}(y) + f_2^{|i}(x, y_{N^{|i,y}}^{|i})t_R^{|i,y}(y) \tag{7.101}$$

$$\begin{aligned}(\mathcal{P}^{|i} f^{|i})(x, y) &= (\Pi^{|i,1}(\Pi^{|i,2} f^{|i}))(x, y) \\ &= \sum_{k=-N^{|i,x}}^{N^{|i,x}} \sum_{l=-N^{|i,y}}^{N^{|i,y}} f^{|i}(x_k^{|i}, y_l^{|i})\omega_k^{|i,x}(x)\omega_l^{|i,y}(y)\end{aligned} \tag{7.102}$$

The approximations for derivatives are defined by direct differentiation; for the x-direction, they are

$$(\Pi_x^{|i,1} f^{|i})(x) = \sum_{k=-N^{|i,x}}^{N^{|i,x}} f_1^{|i}(x_k^{|i}, y)(\omega_k^{|i,x})'(x) \tag{7.103}$$

$$\begin{aligned}(\mathcal{P}_x^{|i} f^{|i})(x, y) &= (\Pi_x^{|i,1}(\Pi^{|i,2} f^{|i}))(x, y) \\ &= \sum_{k=-N^{|i,x}}^{N^{|i,x}} \sum_{l=-N^{|i,y}}^{N^{|i,y}} f^{|i}(x_k^{|i}, y_l^{|i})(\omega_k^{|i,x})'(x)\omega_l^{|i,y}(y),\end{aligned} \tag{7.104}$$

and for the y-direction,

$$(\Pi_y^{|i,2} f^{|i})(y) = \sum_{k=-N^{|i,y}}^{N^{|i,y}} f_2^{|i}(x, y_k^{|i})(\omega_k^{|i,y})'(y) \tag{7.105}$$

$$\begin{aligned}(\mathcal{P}_y^{|i} f^{|i})(x,y) &= (\Pi_y^{|i,2}(\Pi^{|i,1} f^{|i}))(x,y) \\ &= \sum_{k=-N^{|i,x}}^{N^{|i,x}} \sum_{l=-N^{|i,y}}^{N^{|i,y}} f^{|i}(x_k^{|i}, y_l^{|i})\omega_k^{|i,x}(x)(\omega_l^{|i,y})'(y).\end{aligned} \tag{7.106}$$

The forms of the T's are trivial.

The following two lemmas will be used later to prove two-dimensional convergence.

Lemma 7.16 (Projector norm bound) *For any given rectangle $Q^{|i}$ and $f^{|i} \in X_{2\alpha}$,*

$$\|\Pi^{|i,1}\| \equiv \sup_{\substack{x\in\Gamma^{|i} \\ \|f^{|i}\|=1}} \left\{ \left| \sum_{k=-N^{|i,x}}^{N^{|i,x}} f^{|i}(x_k^{|i}, \overline{y}^{|i})\omega_k^{|i,x}(x) \right| \right\} \le C_1 + C_2 \log N^{|i,x}, \tag{7.107}$$

$$\begin{aligned}\left\|\frac{h^{|i,x}}{(\phi^{|i,x})'}\Pi_x^{|i,1}\right\| &\equiv \sup_{\substack{x\in\Gamma^{|i} \\ \|f^{|i}\|=1}} \left\{ \left| \frac{h^{|i,x}}{(\phi^{|i,x})'(x)} \sum_{k=-N^{|i,x}}^{N^{|i,x}} f^{|i}(x_k^{|i}, \overline{y}^{|i})(\omega_k^{|i,x})'(x) \right| \right\} \\ &\le C_1 + C_2 \log N^{|i,x}\end{aligned} \tag{7.108}$$

$$\begin{aligned}\left\|\frac{h^{|i,y}}{(\phi^{|i,y})'}\Pi_y^{|i,2}\right\| &\equiv \sup_{\substack{y\in\Gamma^{|i} \\ \|f^{|i}\|=1}} \left\{ \left| \frac{h^{|i,y}}{(\phi^{|i,y})'(y)} \sum_{k=-N^{|i,y}}^{N^{|i,y}} f^{|i}(\overline{x}^{|i}, y_k^{|i})(\omega_k^{|i,y})'(y) \right| \right\} \\ &\le C_1 + C_2 \log N^{|i,y}\end{aligned} \tag{7.109}$$

Proof: For brevity, let $g(x) = f(x, \overline{y})$, and ignore unnecessary indices; using Equation 7.27 and Lemmas 7.8 and 7.6, get the bound on $\|\Pi^{|,1}\|$ as follows:

$$\begin{aligned}\|\Pi^{|,1}\| = \sup_{\substack{x\in\Gamma \\ \|f\|=1}} \Bigg\{ \Bigg| &\sum_{k=-N+1}^{N-1} \Big[g(x_k) - g(x_{-N})t_L(x_k) - g(x_N)t_R(x_k) \Big] S_k(x) \\ &+ t_L(x)g(x_{-N}) + t_R(x)g(x_N) \\ &- t_L(x_N)S_N(x)g(x_{-N}) - t_R(x_{-N})S_{-N}(x)g(x_N) \Bigg| \Bigg\}\end{aligned} \tag{7.110}$$

$$\begin{aligned}&\le 3 \sup_{x\in\Gamma} \left\{ \sum_{k=-N+1}^{N-1} |S_k(x)| \right\} + 2 + C_1 e^{-\sqrt{\pi d \alpha N}} \\ &\le C_2 + C_3 e^{-\sqrt{\pi d \alpha N}} + C_4 \log N\end{aligned}$$

Equation 7.107 directly follows from this last inequality.

For the x-derivative operator, the sequence is

$$
\begin{aligned}
\|\frac{h^{|i,x}}{(\phi^{|i,x})'}\Pi_x^{|,1}\| = \sup_{\substack{x\in\Gamma \\ \|f\|=1}} \Bigg\{ \Bigg| \sum_{k=-N+1}^{N-1} & \Big[g(x_k) - g(x_{-N})t_L(x_k) - g(x_N)t_R(x_k) \Big] \\
& \cdot S_k'(x)\frac{h^{|i,x}}{(\phi^{|i,x})'(x)} \\
& + t_L'(x)\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}g(x_{-N}) + t_R'(x)\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}g(x_N) \\
& - \frac{h^{|i,x}}{(\phi^{|i,x})'(x)}[t_L(x_N)S_N'(x)g(x_{-N}) \\
& \qquad + t_R(x_{-N})S_{-N}'(x)g(x_N)] \Bigg| \Bigg\}
\end{aligned} \tag{7.111}
$$

$$
\begin{aligned}
&\leq 3\sup_{x\in\Gamma}\left\{\sum_{k=-N+1}^{N-1}|S_k'(x)\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}|\right\} + h^{|i,x}/2 + 2e^{-\sqrt{\pi d\alpha N}} \\
&\leq C_2\log N + C_3/\sqrt{N} + C_4 e^{-\sqrt{\pi d\alpha N}}
\end{aligned}
$$

The bound for $\left\|\frac{h^{|i,y}}{(\phi^{|i,y})'}\Pi_y^{|i,2}\right\|$ is derived similarly. • • •

Lemma 7.17 (Directional interpolation) *For $f \in X_{2\alpha}$ and $(x, y) \in Q^{|i}$,*

$$|f_1^{|i}(x,y) - (\Pi^{|i,1}f^{|i})(x,y)| \leq C_1\sqrt{N^{|i,x}}e^{-\sqrt{\pi d\alpha N^{|i,x}}} \tag{7.112}$$

$$|f_2^{|i}(x,y) - (\Pi^{|i,2}f^{|i})(x,y)| \leq C_2\sqrt{N^{|i,y}}e^{-\sqrt{\pi d\alpha N^{|i,y}}} \tag{7.113}$$

$$|\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}[f_1^{|i}(x,y) - (\Pi^{|i,1}f^{|i})(x,y)]| \leq C_3\sqrt{N^{|i,x}}e^{-\sqrt{\pi d\alpha N^{|i,x}}} \tag{7.114}$$

$$|\frac{h^{|i,y}}{(\phi^{|i,y})'(y)}[f_2^{|i}(x,y) - (\Pi^{|i,2}f^{|i})(x,y)]| \leq C_4\sqrt{N^{|i,y}}e^{-\sqrt{\pi d\alpha N^{|i,y}}} \tag{7.115}$$

Proof: These results follow directly from Definition 7.15 and Theorems 7.9 and 7.10. • • •

The next theorem presents the final results of the preliminary part of this chapter: two-dimensional value- and derivative interpolation.

Theorem 7.18 (Two-dimensional approximation) *Let* $f^{|i} \in X^{|i}_{2\alpha}$ *and choose* $N = \min\{N^{|i,x}, N^{|i,y}\}$. *Then*

$$\begin{aligned}\|f^{|i}(x,y) - (\mathcal{P}^{|i} f^{|i})(x,y)\| &\equiv \sup_{(x,y)\in Q^{|i}} |f^{|i}(x,y) - (\mathcal{P}^{|i} f^{|i})(x,y)| \\ &\leq (C_1 + C_2 \log N)\sqrt{N} e^{-\sqrt{\pi d \alpha N}},\end{aligned} \tag{7.116}$$

$$\|\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}[f^{|i}_x(x,y) - (\mathcal{P}^{|i}_x f^{|i})(x,y)]\| \leq (C_1 + C_2 \log N)\sqrt{N} e^{-\sqrt{\pi d \alpha N}}, \tag{7.117}$$

and

$$\|\frac{h^{|i,y}}{(\phi^{|i,y})'(y)}[f^{|i}_y(x,y) - (\mathcal{P}^{|i}_y f^{|i})(x,y)]\| \leq (C_1 + C_2 \log N)\sqrt{N} e^{-\sqrt{\pi d \alpha N}}. \tag{7.118}$$

Proof: For brevity, drop unnecessary indices. Using the identity

$$f - \mathcal{P} f = f - \Pi_1 f + \Pi_1(f - \Pi_2 f), \tag{7.119}$$

together with Lemmas 7.16 and 7.17, obtain the following sequence.

$$\begin{aligned}&\|f(x,y) - (\mathcal{P} f)(x,y)\| \\ &\qquad \leq \|f(x,y) - (\Pi_1 f)(x,y)\| + \|\Pi_1\| \|f(x,y) - (\Pi_2 f)(x,y)\| \\ &\qquad \leq C_1 \sqrt{N^{|i,x}} e^{-\sqrt{\pi d \alpha N^{|i,x}}} + (C_2 + C_3 \log N^{|i,x})\sqrt{N^{|i,y}} e^{-\sqrt{\pi d \alpha N^{|i,y}}}\end{aligned} \tag{7.120}$$

Equation 7.116 now follows.

Next,

$$\frac{h^{|i,x}}{(\phi^{|i,x})'}[f_x - \mathcal{P}_x f] = \frac{h^{|i,x}}{(\phi^{|i,x})'}[f_x - \Pi^{|,1}_x f] + \frac{h^{|i,x}}{(\phi^{|i,x})'}\Pi^{|,1}_x (f - \Pi^{|,2} f), \tag{7.121}$$

again used with Lemmas 7.16 and 7.17, yields

$$\begin{aligned}&\|\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}[f_x - \mathcal{P}_x f](x,y)\| = \\ &\qquad \|\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}[f_x - \Pi^{|,1}_x f](x,y)\| \\ &\qquad + \|\frac{h^{|i,x}}{(\phi^{|i,x})'(x)}\Pi^{|,1}_x\| \|(f - \Pi^{|,2} f)(x,y)\| \\ &\qquad \leq C_1 \sqrt{N^{|i,x}} e^{-\sqrt{\pi d \alpha N^{|i,x}}} + (C_2 + C_3 \log N^{|i,x})\sqrt{N^{|i,y}} e^{-\sqrt{\pi d \alpha N^{|i,y}}}\end{aligned} \tag{7.122}$$

from which Equation 7.118 follows.

For the y-partial, the identity

$$\frac{h^{|i,y}}{(\phi^{|i,y})'}[f_y - \mathcal{P}_y f] = \frac{h^{|i,y}}{(\phi^{|i,y})'}[f_y - \Pi_y^{|,2} f] + \frac{h^{|i,y}}{(\phi^{|i,y})'}\Pi_y^{|,2}(f - \Pi^{|,1} f), \tag{7.123}$$

can be used in the same way to obtain Equation 7.118. • • •

The next five definition provide the notation and structure for the first major part of this chapter, the setup and solution of a linear algebraic system of equations from a given system of elliptic PDEs.

First, a matrix has only rows and columns, thus two indices. Here, many quantities have to be dealt with, requiring many more indices. The following definition simply requires one-to-one mappings between these sets of indices to exist.

Definition 7.19 ($\mathbb{r}(p,q), \mathbb{c}(l,m)$) *Let $\{i\}$ be an enumeration of all collocation points of the current appropriate region. Denote each point by its index tuple, (p,q), and let $\mathbb{r}(p,q)$ be a one-to-one map such that $i = \mathbb{r}(p,q)^{-1}$.*

Similarly, let $\{j\}$ be an enumeration of all unknown terms of the current appropriate unknown. Denote each term by its index tuple, (l,m), and let $\mathbb{c}(l,m)$ be a one-to-one map such that $j = \mathbb{c}(l,m)^{-1}$.

Next is the specification for forming a discrete vector of values given a function.

Definition 7.20 (Discretized unknowns) *$[u^{n|d}]$ is the vector with entry $\mathbb{c}(l,m)$ given by*

$$[u^{n|d}]_{\mathbb{c}(l,m)} = u^{n|d}(x_l, y_m) \tag{7.124}$$

The given equations and boundary conditions are viewed as a collection of operators acting on the unknowns, with given right-hand sides. All operators in the first-order block systems considered in this work have the following general structure.

Definition 7.21 (Operators) *The operator for the i^{th} unknown in the j^{th} equation for point-region R of rectangle k is written as*

$$L^{i|k}_{j\wedge R} = \left(E^{i|k}_{j\wedge R}(x,y)I + F^{i|k}_{j\wedge R}(x,y)\frac{\partial}{\partial x} + G^{i|k}_{j\wedge R}(x,y)\frac{\partial}{\partial y} \right) \tag{7.125}$$

Zero or more of E, F, G are nonzero.

The notation $[L^{i|k}_{j\wedge R}]$ denotes a rectangular array of numbers with entry $(\mathbb{r}(p,q), \mathbb{c}(l,m))$ given by

$$[L^{i|k}_{j\wedge R}]_{\mathbb{r}(p,q),\mathbb{c}(l,m)} = L^{i|k}_{j\wedge R}\omega^{|j,x}_{l}(x_p)\omega^{|j,y}_{m}(y_q) \tag{7.126}$$

where l and m span the appropriate index ranges.

Analogously, all right-hand-side functions are categorized as follows.

Definition 7.22 (Right-hand functions) *The right-hand function for the j^{th} equation of region R in domain k is written as*

$$f^{|k}_{j\wedge \mathsf{R}}(x,y) \tag{7.127}$$

The vector of values of such a function at sinc points is denoted by $[f^{|k}_{j\wedge \mathsf{R}}]$ and its $\mathbb{r}(p,q)$ entry is given by

$$[f^{|k}_{j\wedge \mathsf{R}}]_{\mathbb{r}(p,q)} = f^{|k}_{j\wedge \mathsf{R}}(x_p, y_q). \tag{7.128}$$

Following is the mathematical structure of the first-order linear algebraic systems considered here. This structure is very complex; as can be seen from Equation 7.126, every individual numeric entry is specified by eight interdependent indices. For examples of this structure, the reader is referred to Chapter 6.

Definition 7.23 (First-order system structure) *Let $Lu = f$ denote the entire first-order linear system to be solved, so that L is a block matrix, with all entries of the form $L^{i|d}_{e\wedge R}$ as defined in Equation 7.125, u is a block vector with entries $u^{i|d}(x,y)$, and f is a block vector with entries $f^{|k}_{j\wedge \mathsf{R}}(x,y)$.*

Further, let $[L]$ denote the block matrix with entries of the form $[L^{i|d}_{e\wedge R}]$ as defined by Equation 7.126, and let $[u]$ denote the block vector with entries $[u^{i|d}]$ as defined by Equation 7.124.

For a system with k unknowns, each block row of $[Lu]$ *is uniquely identified by* $[Lu]_{i\wedge R|d}$*, and the row* $\mathbb{r}(p,q)$ *of each block row has the form*

$$\begin{aligned}([Lu]_{i\wedge R|d})_{\mathbb{r}(p,q)} &= L^{1|d}_{e\wedge R}u^{1|d}(x_p, y_q) + \cdots + L^{k|d}_{e\wedge R}u^{k|d}(x_p, y_q) \\ &\quad + \{L^{1|d_o}_{e\wedge R_o}u^{1|d_o}(x_r, y_s) + \cdots + L^{k|d_o}_{e\wedge R_o}u^{k|d_o}(x_r, y_s)\}\end{aligned} \tag{7.129}$$

where the braced sum contains the overlap blocks from a domain d_o *with boundary* R_o *adjacent to the R boundary of domain d, and the points* (x_p, y_q) *and* (x_r, y_s) *are the sinc collocation points appropriate for the respective regions.*

Analogously, each block row of $[L][u]$ *is uniquely identified by* $([L][u])_{i\wedge R|d}$*, and for a system with k unknowns, each row of such a block row has the form*

$$\begin{aligned}(([L][u])_{i\wedge R|d})_{\mathbb{r}(p,q)} &= ([L^{1|d}_{e\wedge R}][u^{1|d}])_{\mathbb{r}(p,q)} + \cdots + ([L^{k|d}_{e\wedge R}][u^{k|d}])_{\mathbb{r}(p,q)} \\ &\quad + \{([L^{1|d_o}_{e\wedge R_o}][u^{1|d_o}])_{\mathbb{r}(r,s)} + \cdots + ([L^{k|d_o}_{e\wedge R_o}][u^{k|d_o}])_{\mathbb{r}(r,s)}\}\end{aligned} \tag{7.130}$$

For a linear system with the preceding structure, a forward error bound can now be shown. This bounds the difference between the exact values of $[Lu]$ at the sinc points[1] and the approximate values obtained from the product of the approximated operator $[L]$ and the exact values before operator application, $[u]$.

Theorem 7.24 (Collocation forward error) *Let* $Lu = f$ *be a first-order system as in Definition 7.23. For every* $L^{i|d}_{e\wedge R}$*, require*

$$\begin{aligned} E^{i|d}_{e\wedge R}(x,y) &\in \mathbf{H}^\infty(Q^{|d}) \\ F^{i|d}_{e\wedge R}(x,y)(\phi^{|j,x})'(x) &\in \mathbf{H}^\infty(Q^{|d}) \\ G^{i|d}_{e\wedge R}(x,y)(\phi^{|j,y})'(y) &\in \mathbf{H}^\infty(Q^{|d}); \end{aligned}$$

for every $u^{i|d}(x,y)$*, require*

$$u^{i|d}(x,y) \in X_{2\alpha}(Q^{|d}).$$

Choose $N = \min\{N^{|i,x}, N^{|i,y}\}$.

Then

$$\left\| [L][u] - [Lu] \right\| \le CN\log(N)e^{-\sqrt{\pi d\alpha N}} \tag{7.131}$$

[1] The values obtained after application of the exact operator L to the vector of exact functions u.

Proof: Let $\epsilon = \log N\sqrt{N}e^{-\sqrt{\pi d\alpha N}}$. Consider an arbitrary row, $\mathbb{r}(p,q)$, of an arbitrary (equation,region, domain) block row, $e \wedge R|d$. For every unknown i,

$$\begin{aligned}
&|E^{i|d}_{e\wedge R} I u^{i|d}(x_p, y_q) - ([E^{i|d}_{e\wedge R}][u^{i|d}])_{\mathbb{r}(p,q)}| \\
&\qquad = |E^{i|d}_{e\wedge R}(x_p, y_q)[u^{i|d}(x_p, y_q) - (\mathcal{P}^{|d} u^{i|d})(x_p, y_q)]| \\
&\qquad \le C\epsilon
\end{aligned} \tag{7.132}$$

and

$$\begin{aligned}
&|F^{i|d}_{e\wedge R} u^{i|d}_x(x_p, y_q) - ([F^{i|d}_{e\wedge R}][u^{i|d}])_{\mathbb{r}(p,q)}| \\
&\qquad = |F^{i|d}_{e\wedge R}(x_p, y_q)[u^{i|d}_x - (\mathcal{P}^{|d}_x u^{i|d})](x_p, y_q)| \\
&\qquad = |F^{i|d}_{e\wedge R}(x_p, y_q)\frac{(\phi^{|d,x})'(x_p)}{h^{|d,x}} \\
&\qquad\qquad \cdot \frac{h^{|d,x}}{(\phi^{|d,x})'(x_p)}[u^{i|d}_x - (\mathcal{P}^{|d}_x u^{i|d})](x_p, y_q)| \\
&\qquad \le C\sqrt{N}\epsilon.
\end{aligned} \tag{7.133}$$

Similarly,

$$|G^{i|d}_{e\wedge R} u^{i|d}_x(x_p, y_q) - ([G^{i|d}_{e\wedge R}][u^{i|d}])_{\mathbb{r}(p,q)}| \le C\sqrt{N}\epsilon. \tag{7.134}$$

From the definition of $L^{i|d}_{e\wedge R}$ in Equation 7.125,

$$\left| L^{i|d}_{e\wedge R} u^{i|d}(x_p, y_q) - ([L^{i|d}_{e\wedge R}][u^{i|d}])_{\mathbb{r}(p,q)} \right| \le C\sqrt{N}\epsilon \tag{7.135}$$

follows. By subtracting equations 7.129 and 7.130, obtain

$$\left| ([Lu]_{e\wedge R|d})_{\mathbb{r}(p,q)} - (([L][u])_{e\wedge R|d})_{\mathbb{r}(p,q)} \right| \le C\sqrt{N}\epsilon \tag{7.136}$$

Therefore,

$$\left\| [Lu]_{e\wedge R|d} - ([L][u])_{e\wedge R|d} \right\| \le C\sqrt{N}\epsilon \tag{7.137}$$

from which

$$\left\| [L][u] - [Lu] \right\| = \max_{e\wedge R|d} \left\| [Lu]_{e\wedge R|d} - ([L][u])_{e\wedge R|d} \right\| \le C\sqrt{N}\epsilon \tag{7.138}$$

follows. • • •

The next theorem is the key theorem for the approximate solution of elliptic systems using the sinc method. It shows that the approximations obtained by solution of the linear algebraic system accurately approximate the exact values at the collocation points, as long as the inverse matrix norm remains small. This condition is easily tested experimentally, and has held for all problems tested to date.

Theorem 7.25 (Collocation convergence) *Let the assumptions of Theorem 7.24 be satisfied, and let $[c]$ be a computed solution to $[L][c] = [f]$ satisfying*

$$\|[L][c] - [f]\| \leq \epsilon_u \tag{7.139}$$

with ϵ_u proportional to the unit-roundoff error. If $[L]^{-1}$ exists then

$$\|[u] - [c]\| \leq \left\|[L]^{-1}\right\| (CN\log(N)e^{-\sqrt{\pi d\alpha N}} + \epsilon_u) \tag{7.140}$$

Proof: Note that

$$[L]([u] - [c]) = [L][u] - [Lu] + [Lu] - [f] + [f] - [L][c] \tag{7.141}$$

and therefore

$$\begin{aligned} \|[u] - [c]\| \leq \left\|[L]^{-1}\right\| \Big(& \|[L][u] - [Lu]\| \\ & + \|[Lu] - [f]\| + \|[f] - [L][c]\|\Big) \\ \leq \left\|[L]^{-1}\right\| \Big(& CN\log(N)e^{-\sqrt{\pi d\alpha N}} + 0 + \epsilon_u\Big). \end{aligned} \tag{7.142}$$

• • •

In practice, the unit-roundoff term is negligible, and the bound $C\|[L]^{-1}\| N\log(N)e^{-\sqrt{\pi d\alpha N}}$ is used.

These last theorems provide the basis for calculation of both values and derivatives of functions at nonsinc points inside the computational domains. Not only does one get exponential accuracy at a predefined set of points in the computational domain, but using these values, an exponentially accurate interpolant is available as well. Thus, unlike in several other numerical methods, no accuracy is lost when values and derivatives need to be calculated at nongrid points.

Theorem 7.26 (Scaled collocation) *For every domain d and unknown i, let $f^{i|d} \in X_{2\alpha}$, $\delta > 0$, and let $[u]$ be a vector with structure as in Definition 7.23, such that*

$$\left\| [u^{i|d}] - [f^{i|d}] \right\| \le \delta. \tag{7.143}$$

Choose $N = \min\{N^{|i,x}, N^{|i,y}\}$.

Then for any $(x, y) \in Q^{|i}$,

$$\left| [f^{i|d} - \mathcal{P}^{|i} u^{i|d}](x, y) \right| \le e^{-\sqrt{\pi d \alpha N}} C \log N \sqrt{N} + C \log^2 N \delta \tag{7.144}$$

$$\left| \frac{h^{|i,x}}{(\phi^{|i,x})'(x)} [f_x^{i|d} - \mathcal{P}_x^{|i} u^{i|d}](x, y) \right| \le e^{-\sqrt{\pi d \alpha N}} C \log N \sqrt{N} + C \log^2 N \delta \tag{7.145}$$

$$\left| \frac{h^{|i,y}}{(\phi^{|i,y})'(y)} [f_y^{i|d} - \mathcal{P}_y^{|i} u^{i|d}](x, y) \right| \le e^{-\sqrt{\pi d \alpha N}} C \log N \sqrt{N} + C \log^2 N \delta \tag{7.146}$$

Proof: Using Equations 7.116 and 7.107 and ignoring indices,

$$\begin{aligned} |f - \mathcal{P}u| &= |f - \mathcal{P}f + \mathcal{P}f - \mathcal{P}u| \\ &\le |f - \mathcal{P}f| + \left\| \Pi^{|i,1} \right\| \left\| \Pi^{|i,2} \right\| \|f - u\| \\ &\le C_1 \log N \sqrt{N} e^{-\sqrt{\pi d \alpha N}} + C_2 \log N^{|i,x} \log N^{|i,y} \cdot \delta \end{aligned} \tag{7.147}$$

and Equation 7.144 follows. The other equations are simply scaled versions. ●●●

Theorem 7.27 (Collocation) *For $[u]$ computed by the present algorithm, and satisfying the conditions of Theorem 7.25, for all $(x, y) \in Q^{|i}$,*

$$\left| [f^{i|d} - \mathcal{P}^{|i} u^{i|d}](x, y) \right| \le e^{-\sqrt{\pi d \alpha N}} (C \log^3 N \left\| [L]^{-1} \right\| N) \tag{7.148}$$

Further, let

$$[a^{|i,x} < a_s^{|i,x} < b_s^{|i,x} < b^{|i,x}] \tag{7.149}$$

$$[a^{|i,y} < a_s^{|i,y} < b_s^{|i,y} < b^{|i,y}] \tag{7.150}$$

and define $Q_s^{|i}$ by

$$Q_s^{|i} = [a_s^{|i,x}, b_s^{|i,x}] \times [a_s^{|i,y}, b_s^{|i,y}] \tag{7.151}$$

Then for all $(x, y) \in Q_s^{|i}$,

$$\left| [f_x^{i|d} - \mathcal{P}_x^{|i} u^{i|d}](x, y) \right| \le e^{-\sqrt{\pi d \alpha N}} (C \log^3 N \left\| [L]^{-1} \right\| N^{3/2}) \left| (\phi^{|i,x})'(x) \right| \tag{7.152}$$

and

$$\left|[f_y^{i|d} - \mathcal{P}_y^{|i} u^{i|d}](x, y)\right| \leq e^{-\sqrt{\pi d \alpha N}} (C \log^3 N \left\|[L]^{-1}\right\| N^{3/2}) \left|(\phi^{|i,y})'(y)\right| \tag{7.153}$$

Proof: Equation 7.148 follows by substitution from Equation 7.140 into Equation 7.144.

Equation 7.152 follows from substitution of Equation 7.140 into Equation 7.145 and using the definition of $h^{|i,x}$.

Equation 7.153 follows from substitution of Equation 7.140 into Equation 7.146 and using the definition of $h^{|i,y}$. • • •

CHAPTER 8

FUTURE WORK

8.1 Method, algorithm and implementation

8.1.1 Input language improvements

The input data language is already customized for the problem; it remains to turn this into a small programming language — extensible in OCAML — so that all parts of the algorithm can be expressed in it, and so that input can be spread across files in any way desired.

8.1.2 Use of the input language in other solvers

The usual approach taken for providing equations to an algorithm (see e.g., (Houstis, Mitchell, and Rice 1985; Kaufman 1990)) is to give specific names or meanings to *operands*. As operands depend on (and determine) the problem, there is no way for an algorithm to automatically identify them properly. With this approach, individual operands must therefore be explicitly identified, which is impractical. Chapter 4.5.1 presents the typed-operator approach which allows unambiguous algorithmic specification of symbolic input data to PDE solvers in a notation almost identical to standard mathematical notation. Use of this typed-operator algorithm provides a key connection between the mathematics presented Chapter 4 and the practical, readable, maintainable implementation described in Chapter 6, but this input format conversion should work equally well for finite-difference and finite-element algorithms. Perhaps it can be added to one of the free packages as proof-of-concept.

8.1.3 Method efficiency and error estimation

The smallest resulting matrix structure for a two-domain problem is shown in Figure 6.12. With $M = N = 17$, this becomes 29400×29400 with 1,643,388 nonzeros, or 0.19% fill. During execution, the fill reverts to $O(m^4)$, and the work to $O(m^6)$ — clearly unacceptable for solution of larger problems, as evidenced in Section 5.2. Further, multiple solutions are needed for convergence checks. Both of these problems may be addressed simultaneously by using a multigrid-like approach for the SINC-ELLPDE algorithm. A successful algorithm in this style would remove the fill and improve the work to $O(m^4)$, reducing the computation time for two-dimensional problems to minutes, and paving the way for three-dimensional problems.

8.1.4 Programming language issues

Readability of programs is vastly improved by the use of proper programming languages. Especially in research, where program requirements are floating requirements, programming in "standard" programming languages — e.g., C, C++, FORTRAN, Java — is extremely detrimental to quickly writing readable, fast, flexible code.[1] Unfortunately, most computer scientists never implement programs for scientific computing, and most computing scientists never learn better ways of programming, i.e., non-FORTRAN, nonimperative styles.

As a good illustration of the complexity introduced by imperative programming, consider again the main collocation algorithm in Figure 6.9. This figure contains only necessary parts, and is therefore independent of implementation methodology. Adding the accumulation of matrix entries, using an imperative style, yields Figure 6.10, which is *more* complex than the original. The author encountered a problem of similar complexity later on, but wrote a solution in a nearly-pure functional style; adding accumulation of data in this way produced a *simplified* graph. In retrospect, imperative programming style was a poor choice for Figure 6.9; a functional version of this algorithm may be written in the future.

Although object-oriented programming receives much attention, *it is vastly*

[1] Of course, using *existing* libraries written in those languages poses little difficulty, as illustrated by the use of the SUPERLU package.

inferior to functional programming in the implementation of complex mathematical algorithms.[2] Many short examples of this could be shown, but this would not convey the true power of the *combination* of features found in modern, functional-programming languages, especially OCAML, listed in the introduction. Eventually, the current code will be cleaned up and packaged for distribution and made available on the Internet. Perhaps this will expose more computing scientists to functional programming.

8.2 Other problems

The primary goal of this dissertation is the accurate numerical solution of material crack problems; considering the algorithm design and implementation problems and the associated need for testing of functions and programs using numerous test problems, these problems are nothing more than another set of test problems, although more complicated than those typically found in the literature on numerical algorithms.

8.2.1 Boundary layers

Other numerical methods do not handle singularities or boundary layers without "assistance"; sinc methods do. Applying the algorithm to such problems may prove fruitful.

8.2.2 Anisotropic materials

Of more recent interest is the solution of composite-material problems; for these, use of anisotropic material models is useful. Anisotropy requires no additions to the present algorithm, only an enhanced front-end to generate the equations. Problems of this type, as solved in e.g., (Clements 1971; Li and Nemat-Nasser 1990), may be addressed in the future. It is important to note

[2] This is the author's personal experience after writing a large C++ program, a large JAVA program, and the present OCAML program which combines features from both. The JAVA and OCAML programs are both quite readable, but only the OCAML program was a pleasure to write and has retained the most important property of all: extensibility. The C++ program (see (Raymond 1991) for a proper definition of C++), due to the extreme clutter required by the language and the lack of other key features, was dropped in 1996.

that anisotropic materials and nonrectangular coordinate systems can be used by the SINC-ELLPDE algorithm without problems, but the input equations would be substantially more complex.

8.2.3 General geometries and non-Cartesian coordinate systems

The use of mathematical cracks with numerical methods is a historical convenience. For closed-form solutions, this simplifies the solution; for numerical methods, this forces solutions to have singular, rather than steep, solutions — which is bad for convergence and accuracy. For the SINC-ELLPDE method, use of a very sharp ellipse (e.g., a major/minor axis ratio $< 10^{-4}$) would likely produce smoother convergence (compare Figure 2.3 to Figure 5.1), narrow the near-boundary gap to < 0.001 (or 0.1% of a rectangle's length, compare Figure 2.9 and Figure 5.5), and reduce the number of needed rectangles with the proper choice of coordinate system, e.g., elliptic-cylindrical coordinates for a crack. At the same time, numerical values should retain good agreement with those obtained for the mathematical crack — perhaps three to five digits.

In this context, it should be noted that the first-order system used on pages 62–63 is just one way to rewrite a second-order system; especially in elasticity, it is be more practical to use the stress definitions τ^{ij} as the derivative terms instead of $\frac{\partial u^1}{\partial x}$, etc. This would result in a simpler, manageable first-order system when using non-Cartesian coordinate systems.

APPENDIX A

PROGRAM FOR DERIVATION OF EQUATIONS

The MPP[1] program was written by the author to address several shortcomings of the MAPLE computer algebra system, to provide customized, more readable syntax for tensor-based expressions, and to provide both code and pretty-printed/typeset output from a single input, to reduce the errors in printed documentation and program alike.

The ideas relevant in the following programs are the use of modules and automatically-resolved dependencies between equations. The command

```
in-module module code
```

places all `code` given to it inside the named `module`; this avoids naming conflicts. Similarly,

```
anonymous names block
```

replaces all occurrences of `names` in the `block` with globally unique versions; it is similar in purpose to Common Lisp's `gensym`. Lastly, the construct

```
define-dependency target dependencies body
```

is analogous to `Makefile` rules, but for expressions. The `target` expression is produced by the `body`, using only the expressions specified as `dependencies`. Together with its companion command, `make`, this provides a very clean way of re-using already-computed expressions in appropriate contexts — via use of modules. The syntax is

```
make target search-paths.
```

[1] Short for Maple Pre-Processor.

The typesetting of the programs and expressions contained in the following sections was done automatically by MPP; these listings serve several purposes.

First, these listings illustrate advantages and shortcomings of automatic pretty-printing. Instead of (1) working out mathematical expressions using a combination of paper and computer algebra system (CAS), (2) manually converting relevant parts of these equations to another computer language, and (3) manually converting (other) relevant parts of these equations for typesetting, only the CAS needs to be used; the expressions can be automatically converted to any other desired computer language, and typeset output is also directly obtained. The disadvantages are (1) the need to implement a (somewhat complex) conversion program to do the format conversions, and (2) the lack of general algorithms to typeset a combination of mathematics and program. To illustrate these points, only a handful of linebreaks were manually inserted to avoid excessive line lengths. In those places were linebreaks were inserted, the following lines are flush left. The reader may expect some less-than-perfect typesetting, but should keep in mind that *almost no extra work* was required to get all these expressions. The reader should compare this with raw (directly printed) MAPLE or FORTRAN programs, especially when viewing the nested summations so common in tensor analysis.

Second, these listings illustrate one way of dealing with full tensor equations using a CAS, here MAPLE. The steps shown are the rectangular Cartesian expansions of relevant tensor formulas, repeated use of these expansions to form the equations for different regions of different rectangles, and the restructuring of the resulting formulas into the form required by the block-conversion algorithm of Section 6.1.

Third, these listings are an example of a subject-specific front end to the SINC-ELLPDE core. They provide the necessary basis for semiautomatic generation of input equations for more complicated geometries in solid mechanics.

Fourth, these listings illustrate the very substantial difference between theory and its implementation. The original equations from Section 3.1 are only around one page; their algorithmic versions in Section A.1 are about five pages (single spaced). Their use as input to a real program, Section A.2, is about 31 pages.

Specifically, Section A.1 shows actual equations as found in textbooks, e.g., (Green and Zerna 1992), suitably wrapped in an algorithmically-useful form, and Section A.2 shows the full MPP program which utilizes this form of the equations used to provide the isotropic tensor elasticity equations in a format usable by the SINC-ELLPDE algorithm.

A.1 General elasticity equations

The program listing of this section shows the elasticity equations as they appear in the mathematical literature, combined with the algorithmic details necessary to make them useful for machine manipulation. Although they are most likely correct[2], they are not meant for detailed analysis, but rather for illustration. They are, and will remain, very sparsely documented.

in-module :deps:elasticity {

Displacement vector components from physical displacement components.

  define-dependency $D^{\square}$ $\{g_{\square\square}, \overline{D}^{\square}\}$ {

    anonymous i {

      for i **from** 1 **to** 3 **do**

        $D^i := \dfrac{\overline{D}^i}{\sqrt{g_{ii}}}$;

      od ;

    } ;

  } ;

} ;

in-module :deps:elasticity {

Physical displacement components from displacement vector components.

[2] These expressions are directly generated from the same source used to provide input to the numerical algorithm, the solution from which agrees well with known results. Thus, these parts are mutual verifications of each other's correctness. It is of course possible that the LaTeX-backend of MPP has a flaw and produces wrong output in certain circumstances. Also, these expressions have not been tested thoroughly in more interesting coordinate systems, so they may contain errors not affecting their rectangular Cartesian expansion.

define-dependency $\overline{D}^{\square}$ $\{g_{\square\square}, D^{\square}\}$ {
  anonymous i {
    for i **from** 1 **to** 3 **do**
      $\overline{D}^{i} := \sqrt{g_{ii}}D^{i};$
    od ;
  } ;
} ;
} ;

in-module :deps:elasticity {

Main displacement equations. $\vec{v}$ is the unknown, $\vec{D}$ the specified.

define-dependency DisplacementEqns $\{D^{\square}, v^{\square}\}$ {
  anonymous i {
    DisplacementEqns := $[(D^{i} = v^{i})\$i = 1..3];$
  } ;
} ;
} ;

in-module :deps:elasticity {

Covariant derivatives. Here, we only need them for the v^i and $T^i{}_k$ components. Notice that to simplify things to their lowest terms and keep them there, we form the table of the results of the derivatives for a particular v instead of just forming the functions. First, CovDiff1Cont.

define-dependency $v^{\square}{}_{\square}$ $\{v^{\square}, \theta^{\square}, \left\{ {\square \atop \square\ \square} \right\}\}$ {
  anonymous s, r, i {
    for i **from** 1 **to** 3 **do**
      for r **from** 1 **to** 3 **do**
        $v^{r}{}_{i} := \frac{\partial v^{r}}{\partial \theta^{i}} + (\sum_{s=1}^{3} \left\{ {r \atop s\ i} \right\} v^{s});$
      od ;
    od ;
  } ;
} ;

CovDiff2ContCov.

define-dependency $v^{\square}{}_{\square\square}$ $\{\theta^{\square}, \left\{ {\square \atop \square\ \ \square} \right\}, v^{\square}{}_{\square}\}$ {

anonymous i, k, l, m {

for i **from** 1 **to** 3 **do**

for k **from** 1 **to** 3 **do**

for l **from** 1 **to** 3 **do**

$v^{i}{}_{kl} := \dfrac{\partial v^{i}{}_{k}}{\partial \theta^{l}} + (\sum_{m=1}^{3} \left\{ {i \atop m\ \ l} \right\} v^{m}{}_{k}) -$
$(\sum_{m=1}^{3} \left\{ {m \atop k\ \ l} \right\} v^{i}{}_{m});$

od ;

od ;

od ;

} ;

} ;

} ;

in-module :deps:elasticity {

The Navier equation core in terms of the stress tensor σ^{ij}, in the θ system. Note that σ is not expanded further here. The rhs would be the body force term, if present.

define-dependency NavierCoreσ $\{\theta^{\square}, \left\{ {\square \atop \square\ \ \square} \right\}, \sigma^{\square\square}\}$ {

anonymous i, k, m {

NavierCoreσ $:= [((\sum_{i=1}^{3} \dfrac{\partial \sigma^{ik}}{\partial \theta^{i}} + (\sum_{m=1}^{3} \left\{ {i \atop m\ \ i} \right\} \sigma^{mk}) +$
$(\sum_{m=1}^{3} \left\{ {k \atop m\ \ i} \right\} \sigma^{im})))\$k = 1..3];$

} ;

} ;

} ;

in-module :deps:elasticity {

Physical traction tensor from contravariant tensor.

define-dependency $\overline{\tau}^{\square\square}$ $\{\tau^{\square\square}, g_{\square\square}\}$ {

anonymous i, k {

for i **from** 1 **to** 3 **do**
  for k **from** 1 **to** 3 **do**
    $\overline{\tau}^{ik} := \tau^{ik}\sqrt{g_{ii}g_{kk}}$;
  od ;
od ;
$\overline{\tau}^{\square\square}$:= :evalSumDiff(:op($\overline{\tau}^{\square\square}$));
} ;
} ;
} ;

in-module :deps:elasticity {

Traction core equations. $\vec{T}$ are the tensor components of the specified boundary conditions; $\vec{n}$ is the normal vector to the boundary, here expressed via covariant components.

$$T^j = (\sum_{i=1}^{3} \tau^{ij} n_i);$$

define-dependency TractionCore {$\tau^{\square\square}$, $n_\square$} {
  anonymous i, j {
    TractionCore := $[((\sum_{i=1}^{3} \tau^{ij} n_i))\$j = 1..3]$;
  } ;
} ;
} ;

in-module :deps:elasticity {

Navier equation core (nonbody force terms) for interior.

define-dependency NavierCore {$g^{\square\square}$, ν, $v^{\square}{}_{\square\square}$} {
  anonymous s, j, i {
    NavierCore := $[((1-2\nu)(\sum_{j=1}^{3}(\sum_{s=1}^{3} g^{sj} v^{i}{}_{sj})) +$
$(\sum_{j=1}^{3}(\sum_{s=1}^{3} g^{is} v^{j}{}_{js})))\$i = 1..3]$;
  } ;
} ;

} ;

in-module :deps:elasticity {

Metric tensors. Notice that we are assuming the x^i system to be rectangular Cartesian, and the mapping given should be a direct mapping to the system in which the work is done.

define-dependency $g_{\square\square}$ $\{x^{\square}, \theta^{\square}\}$ {

$g_{\square\square}$:= :array$(1 .. 3,\ 1 .. 3)$;

anonymous i, j, m {

for i **from** 1 **to** 3 **do**

for j **from** 1 **to** 3 **do**

$$g_{ij} := \left(\sum_{m=1}^{3} \frac{\partial x^m}{\partial \theta^i}\frac{\partial x^m}{\partial \theta^j}\right);$$

od ;

od ;

} ;

$g_{\square\square}$:= :evalSumDiff(:op($g_{\square\square}$));

} ;

define-dependency $g^{\square\square}$ $\{g_{\square\square}\}$ {

$g^{\square\square}$:= :linalg[:inverse]($g_{\square\square}$);

} ;

Christoffel symbols.

define-dependency $\left[\square, \square\ \square\right]$ $\{g_{\square\square}, \theta^{\square}\}$ {

anonymous i, j, k {

for i **from** 1 **to** 3 **do**

for j **from** 1 **to** 3 **do**

for k **from** 1 **to** 3 **do**

$$\left[i, j\ k\right] := \frac{1}{2}\left(\frac{\partial g_{ij}}{\partial \theta^k} + \frac{\partial g_{ik}}{\partial \theta^j} - \frac{\partial g_{jk}}{\partial \theta^i}\right);$$

od ;

od ;

od ;

$$\left[\square,\square\ \square\right] := \mathsf{:evalSumDiff}\left(\mathsf{:op}\left(\left[\square,\square\ \square\right]\right)\right);$$

} ;

} ;

define-dependency $\left\{ \begin{matrix} \square \\ \square\ \ \square \end{matrix} \right\}$ $\{g^{\square\square}, \left[\square,\square\ \square\right]\}$ {

anonymous i, j, k, l {

for i **from** 1 **to** 3 **do**

for j **from** 1 **to** 3 **do**

for k **from** 1 **to** 3 **do**

$$\left\{ \begin{matrix} i \\ j\ \ k \end{matrix} \right\} := (\sum_{l=1}^{3} g^{il} \left[l, j\ k\right]);$$

od ;

od ;

od ;

$$\left\{ \begin{matrix} \square \\ \square\ \ \square \end{matrix} \right\} := \mathsf{:evalSumDiff}\left(\mathsf{:op}\left(\left\{ \begin{matrix} \square \\ \square\ \ \square \end{matrix} \right\}\right)\right);$$

} ;

} ;

Stress in terms of displacements.

define-dependency $\tau^{\square\square}$ $\{\mu, g^{\square\square}, v^{\square}{}_{\square}, \nu\}$ {

anonymous i, j, s {

for i **from** 1 **to** 3 **do**

for j **from** 1 **to** 3 **do**

$$\tau^{ij} := \mu((\sum_{s=1}^{3} g^{js} v^{i}{}_{s}) + (\sum_{s=1}^{3} g^{is} v^{j}{}_{s}) + (\sum_{s=1}^{3} \frac{2\nu}{(1-2\nu)} g^{ij} v^{s}{}_{s}));$$

od ;

od ;

$$\tau^{\square\square} := \mathsf{:evalSumDiff}\left(\mathsf{:op}\left(\tau^{\square\square}\right)\right);$$

} ;

} ;

Traction vector from physical traction.

define-dependency $T^{\square}$ $\{\overline{T}^{\square}, g_{\square\square}\}$ {
  anonymous i {
    for i **from** 1 **to** 3 **do**
      $T^i := \dfrac{\overline{T}^i}{\sqrt{g_{ii}}};$
    od ;
  } ;
} ;

Unit vector's components from vector's components

define-dependency $n^{\square}$ $\{g_{\square\square}, \overline{n}^{\square}\}$ {
  anonymous i, j, k {
    for i **from** 1 **to** 3 **do**
      $n^i := \dfrac{\overline{n}^i}{\sqrt{(\sum_{j=1}^{3}(\sum_{k=1}^{3} g_{kj}\overline{n}^k\overline{n}^j))}};$
    od ;
    $n^{\square} := \mathsf{:evalSumDiff}(\mathsf{:op}(n^{\square}));$
  } ;
} ;

Covariant components – this "lowering" of indices is really an operator...

define-dependency $n_{\square}$ $\{g_{\square\square}, n^{\square}\}$ {
  anonymous i, j {
    for i **from** 1 **to** 3 **do**
      $n_i := (\sum_{j=1}^{3} g_{ij} n^j);$
    od ;
    $n_{\square} := \mathsf{:evalSumDiff}(\mathsf{:op}(n_{\square}));$
  } ;
} ;
} ;

A.2 Expansion of general equations

The program listing of this section shows the use of the elasticity equations of Section A.1 in the construction of the input structure required by the block-conversion algorithm of Section 6.1.

The rectangular Cartesian expansions of relevant tensor formulas is shown in Section A.2.1, utility functions and expressions are shown in Section A.2.2, general two-dimensional equations and expressions are derived in Section A.2.3, and the two-dimensional expansions are used repeatedly in Sections A.2.4 and A.2.5 to form the equations for different rectangles. Finally, the restructuring of the resulting formulas into the form required by the block-conversion algorithm of Section 6.1 is done in Section A.2.6.

Throughout, italicized text indicates expressions whose expansion is shown in Section B; small-type numbers to the left of program lines indicate the source location of the code.

Although the programs are most likely correct, they contain little error checking; they are meant for competent (MAPLE) programmers with an understanding of the elasticity equations and data structures involved, and are intended to illustrate one possible approach of converting from very loosely specified mathematical equations to a precisely-specified data structure. They are, and will remain, very sparsely documented.

A.2.1 Basic expressions and boundary conditions

448 **in-module** a {

The transformation from target system to rectangular. The θ are the target system components.

451 $x^1 := \theta^1;$

452 $x^2 := \theta^2;$

453 $x^3 := \theta^3;$

} ;

456 **in-module** a {

Get the metric tensor.

458 **define-terminal** $\theta^{\square}$ {

$\theta^{\square}$;

} ;

459 **define-terminal** $x^{\square}$ {

$x^{\square}$;

} ;

Metric tensor g_{ij}

make $g_{\square\square}$:deps:elasticity ;

} ;

466 **in-module** a {

And simplify it as far as possible.

468 :simplify0 := **proc(**foo**)**

:simplify(:evalSumDiff(foo));

end;

Metric tensor g_{ij}

474 $g_{\square\square}$:= :map(:simplify0, $g_{\square\square}$);

} ;

481 **in-module** a {

make $g^{\square\square}$:deps:elasticity ;

} ;

489 **in-module** b {

Get the fully evaluated expressions for the three-dimensional equations.

First, use a simple form of the solution vector v, and "manually" simplify prerequisites.

496 **define-terminal** μ {

μ;

} ;

497 **define-terminal** ν {

ν;

} ;

define-terminal $v^{\square}$ {

$v^1 := u^{\square}$;

$v^2 := u^{\square}$;

$v^3 := u^{\square}$;

} ;

} ;

in-module b {

make $\left[\square, \square\ \square\right]$:deps:elasticity :a;

First Christoffel symbol

$\left[\square, \square\ \square\right]$:= :map(:simplify0, $\left[\square, \square\ \square\right]$);

} ;

in-module b {

make $\left\{ {\square \atop \square\ \ \square} \right\}$:deps:elasticity :a;

Second Christoffel symbol

$\left\{ {\square \atop \square\ \ \square} \right\}$:= :map(:simplify0, $\left\{ {\square \atop \square\ \ \square} \right\}$);

} ;

in-module b {

make $v^{\square}{}_{\square}$:deps:elasticity :a;

$v^{\square}{}_{\square}$:= :map(:simplify0, $v^{\square}{}_{\square}$);

} ;

in-module b {

make $v^{\square}{}_{\square\square}$:deps:elasticity :a;

$v^{\square}{}_{\square\square}$:= :map(:simplify0, $v^{\square}{}_{\square\square}$);

} ;

in-module b1 {

A routine to produce readable output from the generated expressions of u^i .

:readableSubstitution1 := **proc(**foo**)**

:mpCollect(:eval(:subs(:b:u$^{\square}$ = :u^1, :b:u$^{\square}$ = :u^2, :b:u$^{\square}$ = :u^3,

```
                        :b:n = :n, :b:μ = :μ, :b:ν = :ν,
                        :a:θ^1 = :θ^1, :a:θ^2 = :θ^2, :a:θ^3 = :θ^3,
                        :subs(:diff = :Diff, foo))), [:Diff, :θ□, :ν, :μ], []);
        end;
    };
    in-module b {
```

A routine for collection.

```
        :exprCollect2 := proc(foo)
            :mpFactor(:mpCollect(foo, [:sin, :cos, :diff, :Diff, :θ□,
                        :ν, :μ], []));
        end;
    };
    in-module b {
        :evalSum := proc(foo)
            :simplify(:eval(:subs(:Sum = :sum, foo)));
        end;
    };
    in-module b {
```

And a routine for σ terms.

```
        simp4 := proc(foo)
            :eval(:subs(:b:n = :n, :b:μ = :μ, :b:ν = :ν, :b:σ□□ = :σ□□,
                        :a:θ□ = :θ□, foo));
        end;
        factorSigma := proc(foo)
            :mpCollect(foo, [:sigma□□, θ□, :diff, :Diff], []);
        end;
    };
```

A.2.1.1 Navier equations

```
in-module b {
    make NavierCore :deps:elasticity :a;
```

The Navier equation core.

```
    NavierCore := :map(:simplify0, NavierCore);
};
```

A.2.1.2 Stress Tensor

```
in-module b {
    make τ□□ :deps:elasticity :a;
```

The Stress tensor τ^{ij}

```
    τ□□ := :map(:simplify0, τ□□);
};
```

A.2.1.3 Navier equations using σ only

```
in-module b {
```

Get the Navier equations in terms of σ.

```
    define-terminal σ□□ {
        σ□□ := :evaln(σ□□);
    };
    make NavierCoreσ :deps:elasticity :a;
};
in-module b {
```

...and produce a readable form.

The Navier core, σ version.

```
    NavierCoreσ := :map(factorSigma, :map(simp4,
                  :map(:evalSum, NavierCoreσ)));
};
```

A.2.2 Substitutions and additions

A.2.2.1 System

in-module b {

The substitutions/equations to convert the second order equations to first order equations.

$$:\mathrm{systemEqns} := [\frac{\partial{:}\mathrm{u}^1}{\partial{:}\theta^1} = {:}\mathrm{u}^{\square},\ \frac{\partial{:}\mathrm{u}^1}{\partial{:}\theta^2} = {:}\mathrm{u}^{\square},\ \frac{\partial{:}\mathrm{u}^2}{\partial{:}\theta^1} = {:}\mathrm{u}^{\square},\ \frac{\partial{:}\mathrm{u}^2}{\partial{:}\theta^2} = {:}\mathrm{u}^{\square}];$$

The substitutions for the n terms.

:doSystemSubs := **proc**(foo)

:eval(:subs(:op(:systemEqns), foo));

end;

} ;

in-module b {

A custom factorization procedure for the NavierCore.

:exprCollect3 := **proc**(foo)

:mpCollect(foo, [:sin, :cos, :a, :b, :diff, :Diff,

$:\theta^{\square}$, $:\nu$, $:\mu$, :n], []);

end;

A procedure to combine the series substitution with collection and human-readable printing.

:collectτ := **proc**(foo)

:exprCollect3(:readableSubstitution1(foo));

end;

} ;

A.2.2.2 Elimination of third dimension components.

in-module b {

$$:\mathrm{eliminationEqns} := [\frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ \frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ \frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ \frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ \frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ \frac{\partial u^{\square}}{\partial{:}\mathrm{a}{:}\theta^3} = 0,\ u^{\square} = 0];$$

The substitutions for the n terms.

```
        :eliminateZ := proc(foo)
            :eval(:subs(:op(:eliminationEqns), foo));
        end;
    };
    in-module b {
        :remove1stOrderEqns := proc(foo)
            if ( not:type(foo, `=`) ) then
                :ERROR(
                  `Wrong argument type -- expected =:`,
foo);
            fi ;
            thelist□ := :rhs(foo);
            if (:nops(thelist□) <> 6 ) then
                :ERROR(
`Wrong number of terms in argument rhs -- expected 6:`,
                  thelist□);
            fi ;
            :lhs(foo) = [:op(1 .. 2, thelist□)];
        end;
    };
```

All first order systems derived here have structure

```
(list
   (equations                       -> OpEqn
      (lhs
         (sum                       -> OpSum
            (terms
               (* coeff             -> OpProd
                  (unknown
                     (or Diff       -> OpDiff
```

```
                              name))))))   -> name
          (rhs (sum))))
```

and are mapped to the indicated format using the following functions.

```
in-module d {
    :makeOpDiff := proc(term, unknown)
        if (:has(term, unknown) ) then
```

Manual pattern matching…

```
            if (:op(0, term) = :Diff ) then
                depVar := :op(1, term);
                indepVar := :op(2, term);
```

For collocation, we need to know the index of the argument of the function with respect to which the derivative is taken. Here the arguments are $\theta^1,\ \theta^2$.

```
                indepVarIndex := :op(1, indepVar);
                if (indepVarIndex = 1 ) then
                    argIndex := 1;
                elif (indepVarIndex = 2) then
                    argIndex := 2;
                else
                    :ERROR(
'Invalid independent variable (name) found in (term):',
                indepVar, term);
                fi ;
                if (depVar <> unknown ) then
                    :ERROR(
'Internal pattern matching failed on:', term);
                fi ;
                :RETURN(:OpDiff(unknown, argIndex));
            else
```

```
                    :ERROR(
'Unidentified term containing dependent variable:', term);
                fi ;
            else
                :ERROR(
'Term does not contain unknown:', term, unknown);
            fi ;
        end;
    } ;
in-module d {
    :makeOpProd := proc(foo)
        for unknown in :toOpForm:nameList do
```

See if the current unknown shows up anywhere in the examined expression.

```
            if (:has(foo, unknown) ) then
```

Coeff will not handle unknowns inside a Diff or other function call, so we check for this next. Since we are only interested in the case of a diff or Diff of our unknown w.r.t. some other variable, we

1. get a list of all Diffs/diffs in the current expression;
2. make sure there is at most one;
3. dissect the one, if present.

```
                :mvcollist□ := [];
                :mvcoltable□□ := :table([0 = 0]);
                :mpmklist□;
                if (:nops(:mvcollist□) > 1 ) then
                    :ERROR(
'Too many diffs found in subexpression:', foo);
                elif (:nops(:mvcollist□) = 1) then
```

```
                    term := :op(:mvcollist□);
                    ret := :OpProd(:coeff(foo, term),
:makeOpDiff(term, unknown));
                else
                    ret := :OpProd(:coeff(foo, unknown), unknown);
                fi ;
                matchFound := true;
                break;
            fi ;
        od ;
        if (matchFound ) then
            ret;
        else
            :ERROR(
'No name of first matches in second',
:toOpForm:nameList, foo);
        fi ;
    end;
};
in-module d {
    :makeOpSum := proc(currLhs)
        if (:type(currLhs, '+') ) then
            numberOfTerms := :nops(currLhs);
            for termNum from 1 to numberOfTerms do
```

Now separate the unknown from its coefficient.

```
                currTerm := :op(termNum, currLhs);
                currOpProd := :makeOpProd(:eval(:subs(
:toOpForm:subsList, currTerm)));
                if (termNum > 1 ) then
```

```
                currOpProd := :OpSum(cumulativeOpProd, currOpProd);
            fi ;
            cumulativeOpProd := currOpProd;
        od ;
    else
        cumulativeOpProd := :makeOpProd(:eval(:subs(:toOpForm:subsList, currLhs)));
    fi ;
    cumulativeOpProd;
  end;
};
in-module d {
  :makeOpEqn := proc(currEqn)
    currLhs := :lhs(currEqn);
    currRhs := :rhs(currEqn);
    lhsOpSum := :makeOpSum(currLhs);
    :OpEqn(lhsOpSum, currRhs);
  end;
};
in-module d {
  :makeOpEqnList := proc(eqnList)
    numberOfEquations := :nops(eqnList);
    newList := [];
    for eqnNum from 1 to numberOfEquations do
        currEqn := :op(eqnNum, eqnList);
        opEqn := :makeOpEqn(currEqn);
        newList := [:op(newList), opEqn];
    od ;
    newList;
```

```
        end;
    } ;
954 in-module toOpForm {
```

Separating the unknown from its coefficient requires the name of the unknown in the current term. Also, the use of indexed types for the unknowns was useful for previous Maple operations, but now simple names are needed.

To handle both requirements, we first substitute the simple names; this will also give a list of said names, with which we can determine the current term's name.

```
964     subsList := [:u□ = :u11, :u□ = :u12, :u□ = :u21, :u□ = :u22,
:u¹ = :u1, :u² = :u2];
975     nameList := [:u1, :u2, :u11, :u12, :u21, :u22];
    } ;
```

A.2.3 General expressions

A.2.3.1 Navier equations for all interiors

```
1001 in-module b {
        NavierCore[1];
        :lprint(NavierCore[1]);
        :eliminateZ(NavierCore[1]);
    } ;
1007 in-module b {
```

Make the series substitutions in the Navier equations and get the reduced main equations. The first two expressions seem to produce the same output, but the second is better collected...

```
        :readableSubstitution1(:exprCollect3(:eliminateZ(NavierCore[1])));
        :exprCollect3(:readableSubstitution1(:eliminateZ(NavierCore[1])));
    } ;
```

```
in-module b {
```

Navier core equations.

Make the simplified global form available. Notice that the third equation is removed.

```
    NavierCoreSeries := [:op(1 .. 2, :map(:exprCollect3,
:map(:readableSubstitution1, :map(:eliminateZ, NavierCore))))];
  } ;
in-module b {
```

Given 2 lists of expressions, return a single list of equations.

```
    :mergeLhsRhs := proc(lhsList, rhsList)
        if (:nops(lhsList) <> :nops(rhsList) ) then
            :ERROR(`list lengths do not match`);
        fi ;
        ret := [];
        for i from 1 to :nops(lhsList) do
            ret := [:op(ret), :op(i, lhsList) = :op(i, rhsList)];
        od ;
    end;
    :rhsToLhs := proc(foo)
        :lhs(foo) − :rhs(foo) = 0;
    end;
  } ;
in-module b {
```

Substitute the first-order system equations. This gives the main equation's lhs expressions; since all domains here have homogeneous right hand sides, the final system of *equations* is also formed here.

The substituted equations form the rest of the system.

```
    NavierCoreSysExpressions := :map(:doSystemSubs,
NavierCoreSeries);
```

First order system equations.

```
        NavierCoreSysEqns := [:op(
:mergeLhsRhs(NavierCoreSysExpressions, [0, 0])),
        :op(:map(:rhsToLhs, :systemEqns))];
    };
    in-module d {
```

Get the `Interior=[...]` portion of the `eqn_input` file format.

```
        InteriorEqn := :Interior = :makeOpEqnList(:b:NavierCoreSysEqns);
    };
```

Since the series substitutions involve only the v^i, nothing needs to be done for the σ version of the Navier equations.

A.2.3.2 Traction equations using stress tensor

```
    in-module b {
```

Eliminate the z-direction components of the stress tensor and collect and simplify the results.

```
        tmp := :op(τ□□);
        τ□□ := :table();
        anonymous i, j {
            for i from 1 to 2 do
                for j from 1 to 2 do
                    τ^ij := :eliminateZ(tmp[i, j]);
                od ;
            od ;
        };
    };
    in-module b {
```

τ^{ij} after series substitutions.

```
        :map(:collectτ, τ□□);
    };
```

A.2.3.3 Displacement vector

Make the series substitutions for the displacement vector and collect and simplify the results.

in-module b {

v^i after series substitutions.

:map(:collectτ, $[(v^i)\$i = 1..3]$);

} ;

A.2.4 Conditions for domain 1

in-module d1Top {

make $v^\square$:deps:elasticity :a, :b;

make $\theta^\square$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[v^1,\ v^2]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

} ;

A.2.4.1 Top conditions

in-module d1Top {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

} ;

in-module d1Top {

SysExpressions := :map(:doSystemSubs, :map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)), :op(:map(:rhsToLhs, :systemEqns))];

};

in-module d1Top {

Get the `:Top =[...]` portion of the `eqn_input`.

Eqn := :Top = :makeOpEqnList(SysEqns);

};

in-module d1Left {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[\frac{\partial v^2}{\partial \theta^1},\ v^1]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

};

A.2.4.2 Left conditions

in-module d1Left {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d1Left {

SysExpressions := :map(:doSystemSubs,
:map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)),
:op(:map(:rhsToLhs, :systemEqns))];

};

in-module d1Left {

Get the `:Left =[...]` portion of the `eqn_input`.

Eqn := :Left = :makeOpEqnList(SysEqns);

};

```
in-module d1Bottom {
    define-terminal n̄^□ {
        n̄^1 := 0;
        n̄^2 := −1;
        n̄^3 := 0;
     } ;
    make n^□ :deps:elasticity :b, :a;
    n^□ := :map(:simplify0, n^□);
} ;
in-module d1Bottom {
    make n_□ :deps:elasticity :b, :a;
    n_□ := :map(:simplify0, n_□);
} ;
in-module d1Bottom {
    make TractionCore :deps:elasticity :b, :a;
    TractionCore := [:op(1 .. 2, :map(:simplify0, TractionCore))];
} ;
in-module d1Bottom {
```

The traction equation core $\tau^{ij}n_i$.

```
    :map(:readableSubstitution1, :map(:exprCollect2, TractionCore));
```

The traction equation rhs.

```
    TractionRhs := [0, :σ];
} ;
```

A.2.4.3 Bottom conditions

```
in-module d1Bottom {
    TractionSysExpressions := :map(:collectτ, TractionCore);
```

The first order traction equation system.

```
    TractionSysEqns := [:op(:mergeLhsRhs(TractionSysExpressions,
TractionRhs)), :op(:map(:rhsToLhs, :systemEqns))];
```

};

in-module d1Bottom {

Get the `:Bottom =[...]` portion of the `eqn_input`, n and non-n equations.

Eqn := :Bottom = :makeOpEqnList(TractionSysEqns);

};

in-module d1Right {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[\frac{\partial v^1}{\partial \theta^1}, \frac{\partial v^2}{\partial \theta^1}]$;

Boundary conditions, rhs.

conditionsRhs := $[0, 0]$;

};

A.2.4.4 Right conditions

in-module d1Right {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d1Right {

SysExpressions := :map(:doSystemSubs, :map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)), :op(:map(:rhsToLhs, :systemEqns))];

};

in-module d1Right {

Get the `:Right =[...]` portion of the `eqn_input`.

Eqn := :Right = :makeOpEqnList(SysEqns);

};

in-module d1RightOL {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[v^1,\ v^2]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

};

A.2.4.5 RightOL conditions

in-module d1RightOL {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d1RightOL {

SysExpressions := :map(:doSystemSubs,
:map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)),
:op(:map(:rhsToLhs, :systemEqns))];

};

in-module d1RightOL {

Get the `:RightOL =[...]` portion of the `eqn_input`.

Eqn := :RightOL = :makeOpEqnList(SysEqns);

};

A.2.5 Conditions for domain 2

in-module d2Right {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[v^1,\ v^2]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

} ;

A.2.5.1 Right conditions

in-module d2Right {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

} ;

in-module d2Right {

SysExpressions := :map(:doSystemSubs,
:map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)),
:op(:map(:rhsToLhs, :systemEqns))];

} ;

in-module d2Right {

Get the `:Right =[...]` portion of the `eqn_input`.

Eqn := :Right = :makeOpEqnList(SysEqns);

} ;

in-module d2Top {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[v^1,\ v^2]$;

Boundary conditions, rhs.

```
conditionsRhs := [0, 0];
};
```

A.2.5.2 Top conditions

```
in-module d2Top {
```

Conditions after simplification and factorization.

```
:map(:collectτ, conditions);
};
in-module d2Top {
SysExpressions := :map(:doSystemSubs, :map(:collectτ,
conditions));
```

The first order equation system.

```
SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)),
:op(:map(:rhsToLhs, :systemEqns))];
};
in-module d2Top {
```

Get the `:Top =[...]` portion of the `eqn_input`.

```
Eqn := :Top = :makeOpEqnList(SysEqns);
};
in-module d2Left {
make v□ :deps:elasticity :a, :b;
make θ□ :deps:elasticity :a, :b;
```

Boundary conditions.

```
conditions := [ -v^1, -v^2];
```

Boundary conditions, rhs.

```
conditionsRhs := [0, 0];
};
```

A.2.5.3 Left conditions

in-module d2Left {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d2Left {

SysExpressions := :map(:doSystemSubs, :map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)), :op(:map(:rhsToLhs, :systemEqns))];

};

in-module d2Left {

Get the `:Left =[...]` portion of the `eqn_input`.

Eqn := :Left = :makeOpEqnList(SysEqns);

};

in-module d2LeftOL {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[\, -\dfrac{\partial v^1}{\partial \theta^1},\ \ -\dfrac{\partial v^2}{\partial \theta^1}]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

};

A.2.5.4 LeftOL conditions

in-module d2LeftOL {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d2LeftOL {

SysExpressions := :map(:doSystemSubs,
:map(:collectτ, conditions));

The first order equation system.

SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)),
:op(:map(:rhsToLhs, :systemEqns))];

};

in-module d2LeftOL {

Get the `:LeftOL =[...]` portion of the `eqn_input`.

Eqn := :LeftOL = :makeOpEqnList(SysEqns);

};

in-module d2Bottom {

make $v^{\square}$:deps:elasticity :a, :b;

make $\theta^{\square}$:deps:elasticity :a, :b;

Boundary conditions.

conditions := $[\frac{\partial v^1}{\partial \theta^2},\ v^2]$;

Boundary conditions, rhs.

conditionsRhs := $[0,\ 0]$;

};

A.2.5.5 Bottom conditions

in-module d2Bottom {

Conditions after simplification and factorization.

:map(:collectτ, conditions);

};

in-module d2Bottom {

SysExpressions := :map(:doSystemSubs,
:map(:collectτ, conditions));

The first order equation system.

1808 SysEqns := [:op(:mergeLhsRhs(SysExpressions, conditionsRhs)), :op(:map(:rhsToLhs, :systemEqns))];

};

1815 **in-module** d2Bottom {

Get the `:Bottom =[...]` portion of the `eqn_input`.

1819 Eqn := :Bottom = :makeOpEqnList(SysEqns);

};

A.2.6 Structure output

For numerics, we use the first order systems converted to Operator format (derived above), arranged appropriately for use by `eqn_read.ml`.

1831 **in-module** eqnInp {

The equations:

1833 structure := :subs(:θ^1 = :x, :θ^2 = :y, :equationspecs$_\square$ = [:unknowns = :toOpForm:nameList, :domains = [
1 = [:regions = [:d:InteriorEqn, :d1Bottom:Eqn, :d1Left:Eqn, :d1Top:Eqn, :d1Right:Eqn, :remove1stOrderEqns(:d1RightOL:Eqn)]],
2 = [:regions = [:remove1stOrderEqns(:d2LeftOL:Eqn), :d2Left:Eqn, :d2Top:Eqn, :d2Right:Eqn, :d:InteriorEqn, :d2Bottom:Eqn]]]]);

APPENDIX B

OUTPUT FROM PROGRAM FOR DERIVATION OF EQUATIONS

This chapter presents the output resulting from the italicized code of Section A.2. As before, note that all of these expressions are computer-generated from the same input which generates the actual code. Thus, the expressions typeset here are really pretty-printed computer program fragments, *not* manually-typed mathematics. As for any program listing, manual adjustment is therefore avoided — it would add no content, and waste time.

B.1 Basic expressions and boundary conditions

Metric tensor g_{ij}

$$\begin{aligned}
[[1,2] &= 0,\\
[2,2] &= 1,\\
[1,3] &= 0,\\
[1,1] &= 1,\\
[3,2] &= 0,\\
[2,3] &= 0,\\
[2,1] &= 0,\\
[3,3] &= 1,\\
[3,1] &= 0]
\end{aligned} \tag{B.1}$$

Metric tensor g^{ij}

$$
\begin{aligned}
[[1,2] &= 0,\\
[2,2] &= 1,\\
[1,3] &= 0,\\
[1,1] &= 1,\\
[3,2] &= 0,\\
[2,3] &= 0,\\
[2,1] &= 0,\\
[3,3] &= 1,\\
[3,1] &= 0]
\end{aligned} \tag{B.2}
$$

First Christoffel symbol

$$
\begin{aligned}
[[2,1,1] &= 0,\\
[3,3,2] &= 0,\\
[2,3,1] &= 0,\\
[3,1,2] &= 0,\\
[3,3,3] &= 0,\\
[3,1,1] &= 0,\\
[3,1,3] &= 0,\\
[1,2,1] &= 0,\\
[3,2,3] &= 0,\\
[1,3,1] &= 0,\\
[3,2,2] &= 0,\\
[3,2,1] &= 0,\\
[1,1,1] &= 0,\\
[2,2,2] &= 0,\\
[3,3,1] &= 0,\\
[1,3,2] &= 0,\\
[1,2,3] &= 0,\\
[1,3,3] &= 0,\\
[1,1,3] &= 0,\\
[1,2,2] &= 0,\\
[2,1,2] &= 0,\\
[2,2,1] &= 0,\\
[2,3,2] &= 0,\\
[2,3,3] &= 0,\\
[2,1,3] &= 0,\\
[1,1,2] &= 0,\\
[2,2,3] &= 0]
\end{aligned} \tag{B.3}
$$

Second Christoffel symbol

$$
\begin{aligned}
[[2,1,1] &= 0,\\
[3,3,2] &= 0,\\
[2,3,1] &= 0,\\
[3,1,2] &= 0,\\
[3,3,3] &= 0,\\
[3,1,1] &= 0,\\
[3,1,3] &= 0,\\
[1,2,1] &= 0,\\
[3,2,3] &= 0,\\
[1,3,1] &= 0,\\
[3,2,2] &= 0,\\
[3,2,1] &= 0,\\
[1,1,1] &= 0,\\
[2,2,2] &= 0,\\
[3,3,1] &= 0,\\
[1,3,2] &= 0,\\
[1,2,3] &= 0,\\
[1,3,3] &= 0,\\
[1,1,3] &= 0,\\
[1,2,2] &= 0,\\
[2,1,2] &= 0,\\
[2,2,1] &= 0,\\
[2,3,2] &= 0,\\
[2,3,3] &= 0,\\
[2,1,3] &= 0,\\
[1,1,2] &= 0,\\
[2,2,3] &= 0]
\end{aligned}
\tag{B.4}
$$

B.1.1 Navier equations

The Navier equation core.

$$
\begin{aligned}
\Bigg[&(2-2\nu)\frac{\partial^2 \mathsf{u}^1}{(\partial\theta^1)^2} + (1-2\nu)\frac{\partial^2 \mathsf{u}^1}{(\partial\theta^2)^2} + (1-2\nu)\frac{\partial^2 \mathsf{u}^1}{(\partial\theta^3)^2} + \frac{\partial^2 \mathsf{u}^3}{\partial\theta^1\partial\theta^3} + \frac{\partial^2 \mathsf{u}^2}{\partial\theta^1\partial\theta^2},\\
&(1-2\nu)\frac{\partial^2 \mathsf{u}^2}{(\partial\theta^1)^2} + (2-2\nu)\frac{\partial^2 \mathsf{u}^2}{(\partial\theta^2)^2} + (1-2\nu)\frac{\partial^2 \mathsf{u}^2}{(\partial\theta^3)^2} + \frac{\partial^2 \mathsf{u}^3}{\partial\theta^2\partial\theta^3} + \frac{\partial^2 \mathsf{u}^1}{\partial\theta^1\partial\theta^2},\\
&(1-2\nu)\frac{\partial^2 \mathsf{u}^3}{(\partial\theta^1)^2} + (1-2\nu)\frac{\partial^2 \mathsf{u}^3}{(\partial\theta^2)^2} + (2-2\nu)\frac{\partial^2 \mathsf{u}^3}{(\partial\theta^3)^2} + \frac{\partial^2 \mathsf{u}^2}{\partial\theta^2\partial\theta^3} + \frac{\partial^2 \mathsf{u}^1}{\partial\theta^1\partial\theta^3}\Bigg]
\end{aligned}
\tag{B.5}
$$

B.1.2 Stress Tensor

The Stress tensor τ^{ij}

$$
\begin{aligned}
[1,2] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^2} + \mu\frac{\partial \mathsf{u}^2}{\partial\theta^1} \\
[2,2] &= \frac{2\mu\left(-1+\nu\right)}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial\theta^1} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^3}{\partial\theta^3} \\
[1,3] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^3} + \mu\frac{\partial \mathsf{u}^3}{\partial\theta^1} \\
[1,1] &= \frac{2\mu\left(-1+\nu\right)}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial\theta^1} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^3}{\partial\theta^3} \\
[3,2] &= \mu\frac{\partial \mathsf{u}^2}{\partial\theta^3} + \mu\frac{\partial \mathsf{u}^3}{\partial\theta^2} \\
[2,3] &= \mu\frac{\partial \mathsf{u}^2}{\partial\theta^3} + \mu\frac{\partial \mathsf{u}^3}{\partial\theta^2} \\
[2,1] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^2} + \mu\frac{\partial \mathsf{u}^2}{\partial\theta^1} \\
[3,3] &= \frac{2\mu\left(-1+\nu\right)}{-1+2\nu}\frac{\partial \mathsf{u}^3}{\partial\theta^3} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial\theta^1} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2} \\
[3,1] &= \mu\frac{\partial \mathsf{u}^1}{\partial\theta^3} + \mu\frac{\partial \mathsf{u}^3}{\partial\theta^1}
\end{aligned} \tag{B.6}
$$

B.1.3 Navier equations using σ only

The Navier core, σ version.

$$
\left[\frac{\partial\sigma^{1,1}}{\partial\theta^1} + \frac{\partial\sigma^{2,1}}{\partial\theta^2} + \frac{\partial\sigma^{3,1}}{\partial\theta^3}, \frac{\partial\sigma^{1,2}}{\partial\theta^1} + \frac{\partial\sigma^{2,2}}{\partial\theta^2} + \frac{\partial\sigma^{3,2}}{\partial\theta^3}, \frac{\partial\sigma^{1,3}}{\partial\theta^1} + \frac{\partial\sigma^{2,3}}{\partial\theta^2} + \frac{\partial\sigma^{3,3}}{\partial\theta^3}\right] \tag{B.7}
$$

B.2 Substitutions and additions

B.2.1 System

The additional/substitution equations.

$$
\left[\frac{\partial \mathsf{u}^1}{\partial\theta^1} = \mathsf{u}\hat{}\,[1,1]\,, \frac{\partial \mathsf{u}^1}{\partial\theta^2} = \mathsf{u}\hat{}\,[1,2]\,, \frac{\partial \mathsf{u}^2}{\partial\theta^1} = \mathsf{u}\hat{}\,[2,1]\,, \frac{\partial \mathsf{u}^2}{\partial\theta^2} = \mathsf{u}\hat{}\,[2,2]\right] \tag{B.8}
$$

B.2.2 Elimination of third dimension components.

The substitutions used (inert form).

$$\left[\frac{\partial \mathsf{u}^1\left(\theta^1,\theta^2,\theta^3\right)}{\partial\theta^3}=0,\frac{\partial^2 \mathsf{u}^1\left(\theta^1,\theta^2,\theta^3\right)}{(\partial\theta^3)^2}=0,\frac{\partial \mathsf{u}^2\left(\theta^1,\theta^2,\theta^3\right)}{\partial\theta^3}=0,\right.$$
$$\frac{\partial^2 \mathsf{u}^2\left(\theta^1,\theta^2,\theta^3\right)}{(\partial\theta^3)^2}=0,\frac{\partial \mathsf{u}^3\left(\theta^1,\theta^2,\theta^3\right)}{\partial\theta^3}=0,\frac{\partial^2 \mathsf{u}^3\left(\theta^1,\theta^2,\theta^3\right)}{(\partial\theta^3)^2}=0, \tag{B.9}$$
$$\left.\mathsf{u}^3\left(\theta^1,\theta^2,\theta^3\right)=0\right]$$

B.3 General expressions

B.3.1 Navier equations for all interiors

Navier core equations.

$$\begin{aligned}(-2\nu+2)\frac{\partial^2\mathsf{u}^1}{(\partial\theta^1)^2}+(1-2\nu)\frac{\partial^2\mathsf{u}^1}{(\partial\theta^2)^2}+\frac{\partial^2\mathsf{u}^2}{\partial\theta^1\partial\theta^2}=0\\ (1-2\nu)\frac{\partial^2\mathsf{u}^2}{(\partial\theta^1)^2}+(-2\nu+2)\frac{\partial^2\mathsf{u}^2}{(\partial\theta^2)^2}+\frac{\partial^2\mathsf{u}^1}{\partial\theta^1\partial\theta^2}=0\end{aligned} \tag{B.10}$$

First-order system equations.

$$\begin{aligned}(-2\nu+2)\frac{\partial\mathsf{u}^{1,1}}{\partial\theta^1}+(1-2\nu)\frac{\partial\mathsf{u}^{1,2}}{\partial\theta^2}+\frac{\partial\mathsf{u}^{2,1}}{\partial\theta^2}=0,\\ (1-2\nu)\frac{\partial\mathsf{u}^{2,1}}{\partial\theta^1}+(-2\nu+2)\frac{\partial\mathsf{u}^{2,2}}{\partial\theta^2}+\frac{\partial\mathsf{u}^{1,1}}{\partial\theta^2}=0\\ \frac{\partial\mathsf{u}^1}{\partial\theta^1}-\mathsf{u}^{1,1}=0\\ \frac{\partial\mathsf{u}^1}{\partial\theta^2}-\mathsf{u}^{1,2}=0\\ \frac{\partial\mathsf{u}^2}{\partial\theta^1}-\mathsf{u}^{2,1}=0\\ \frac{\partial\mathsf{u}^2}{\partial\theta^2}-\mathsf{u}^{2,2}=0\end{aligned} \tag{B.11}$$

B.3.2 Traction equations using stress tensor

τ^{ij} after series substitutions.

$$
\begin{aligned}
[1,2] &= \mu \frac{\partial \mathsf{u}^1}{\partial \theta^2} + \mu \frac{\partial \mathsf{u}^2}{\partial \theta^1} \\
[2,2] &= \frac{-2\mu\,(1-\nu)}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial \theta^2} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial \theta^1} \\
[1,1] &= \frac{2\mu\,(-1+\nu)}{-1+2\nu}\frac{\partial \mathsf{u}^1}{\partial \theta^1} - \frac{2\mu\nu}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial \theta^2} \\
[2,1] &= \mu \frac{\partial \mathsf{u}^1}{\partial \theta^2} + \mu \frac{\partial \mathsf{u}^2}{\partial \theta^1}
\end{aligned}
\tag{B.12}
$$

B.3.3 Displacement vector

v^i after series substitutions.

$$\left[\mathsf{u}^1, \mathsf{u}^2, \mathsf{u}^3\right] \tag{B.13}$$

B.4 Conditions for domain 1

B.4.1 Top conditions

Starting conditions.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.14}$$

Starting rhs.

$$[0,0] \tag{B.15}$$

Conditions after simplification and factorization.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.16}$$

The first-order equation system.

$$
\begin{aligned}
\Big[\mathsf{u}^1 = 0, \mathsf{u}^2 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0,& \\
\frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Big]&
\end{aligned}
\tag{B.17}
$$

B.4.2 Left conditions

Starting conditions.

$$\left[\frac{\partial \mathsf{u}^2}{\partial \theta^1}, \mathsf{u}^1\right] \tag{B.18}$$

Starting rhs.

$$[0, 0] \tag{B.19}$$

Conditions after simplification and factorization.

$$\left[\frac{\partial \mathsf{u}^2}{\partial \theta^1}, \mathsf{u}^1\right] \tag{B.20}$$

The first-order equation system.

$$\left[\mathsf{u\hat{}}\,[2,1] = 0, \mathsf{u}^1 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u\hat{}}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u\hat{}}\,[1,2] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u\hat{}}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u\hat{}}\,[2,2] = 0\right] \tag{B.21}$$

B.4.3 Bottom conditions

The traction equation core $\tau^{ij} n_i$.

$$\left[-\mu \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mu \frac{\partial \mathsf{u}^2}{\partial \theta^1}, \frac{-2\mu(-1+\nu)}{-1+2\nu} \frac{\partial \mathsf{u}^2}{\partial \theta^2} + \frac{2\mu\nu \frac{\partial \mathsf{u}^1}{\partial \theta^1}}{-1+2\nu}\right] \tag{B.22}$$

The calculated surface normal n_i.

$$\mathsf{table}\,([1 = 0, 2 = -1, 3 = 0]) \tag{B.23}$$

The traction equation core rhs.

$$[0, \sigma] \tag{B.24}$$

The first-order traction equation system.

$$\left[-\mu \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mu \frac{\partial \mathsf{u}^2}{\partial \theta^1} = 0, \frac{-2\mu(-1+\nu)}{-1+2\nu} \frac{\partial \mathsf{u}^2}{\partial \theta^2} + \frac{2\mu\nu \frac{\partial \mathsf{u}^1}{\partial \theta^1}}{-1+2\nu} = \sigma, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u\hat{}}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u\hat{}}\,[1,2] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u\hat{}}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u\hat{}}\,[2,2] = 0\right] \tag{B.25}$$

B.4.4 Right conditions

Starting conditions.

$$\left[\frac{\partial \mathsf{u}^1}{\partial \theta^1}, \frac{\partial \mathsf{u}^2}{\partial \theta^1}\right] \tag{B.26}$$

Starting rhs.

$$[0, 0] \tag{B.27}$$

Conditions after simplification and factorization.

$$\left[\frac{\partial \mathsf{u}^1}{\partial \theta^1}, \frac{\partial \mathsf{u}^2}{\partial \theta^1}\right] \tag{B.28}$$

The first-order equation system.

$$\begin{split}\Big[\mathsf{u}\hat{}\,[1,1] = 0, \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} \\ - \mathsf{u}\hat{}\,[1,2] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Big]\end{split} \tag{B.29}$$

B.4.5 RightOL conditions

Starting conditions.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.30}$$

Starting rhs.

$$[0, 0] \tag{B.31}$$

Conditions after simplification and factorization.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.32}$$

The first-order equation system.

$$\begin{split}\Big[\mathsf{u}^1 = 0, \mathsf{u}^2 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0, \\ \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Big]\end{split} \tag{B.33}$$

B.5 Conditions for domain 2

B.5.1 Right conditions

Starting conditions.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.34}$$

Starting rhs.

$$[0, 0] \tag{B.35}$$

Conditions after simplification and factorization.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.36}$$

The first-order equation system.

$$\begin{aligned}\Bigg[\mathsf{u}^1 = 0, \mathsf{u}^2 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0,\\ \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Bigg]\end{aligned} \tag{B.37}$$

B.5.2 Top conditions

Starting conditions.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.38}$$

Starting rhs.

$$[0, 0] \tag{B.39}$$

Conditions after simplification and factorization.

$$\left[\mathsf{u}^1, \mathsf{u}^2\right] \tag{B.40}$$

The first-order equation system.

$$\begin{aligned}\Bigg[\mathsf{u}^1 = 0, \mathsf{u}^2 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0,\\ \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Bigg]\end{aligned} \tag{B.41}$$

B.5.3 Left conditions

Starting conditions.

$$\left[-\mathsf{u}^1, -\mathsf{u}^2\right] \tag{B.42}$$

Starting rhs.

$$[0, 0] \tag{B.43}$$

Conditions after simplification and factorization.

$$\left[-\mathsf{u}^1, -\mathsf{u}^2\right] \tag{B.44}$$

The first-order equation system.

$$\begin{aligned}\Big[-\mathsf{u}^1 = 0, -\mathsf{u}^2 = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0,\\ \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Big]\end{aligned} \tag{B.45}$$

B.5.4 LeftOL conditions

Starting conditions.

$$\left[-\frac{\partial \mathsf{u}^1}{\partial \theta^1}, -\frac{\partial \mathsf{u}^2}{\partial \theta^1}\right] \tag{B.46}$$

Starting rhs.

$$[0, 0] \tag{B.47}$$

Conditions after simplification and factorization.

$$\left[-\frac{\partial \mathsf{u}^1}{\partial \theta^1}, -\frac{\partial \mathsf{u}^2}{\partial \theta^1}\right] \tag{B.48}$$

The first-order equation system.

$$\begin{aligned}\Big[-\mathsf{u}\hat{}\,[1,1] = 0, -\mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^1}{\partial \theta^1} - \mathsf{u}\hat{}\,[1,1] = 0,\\ \frac{\partial \mathsf{u}^1}{\partial \theta^2} - \mathsf{u}\hat{}\,[1,2] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^1} - \mathsf{u}\hat{}\,[2,1] = 0, \frac{\partial \mathsf{u}^2}{\partial \theta^2} - \mathsf{u}\hat{}\,[2,2] = 0\Big]\end{aligned} \tag{B.49}$$

B.5.5 Bottom conditions

Starting conditions.

$$\left[-\mu\frac{\partial \mathsf{u}^1}{\partial\theta^2}-\mu\frac{\partial \mathsf{u}^2}{\partial\theta^1},\frac{-2\mu\left(-1+\nu\right)}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2}+\frac{2\mu\nu\frac{\partial \mathsf{u}^1}{\partial\theta^1}}{-1+2\nu}\right] \tag{B.50}$$

Starting rhs.

$$\mathsf{table}\left(\left[1=0,2=-1,3=0\right]\right) \tag{B.51}$$

Conditions after simplification and factorization.

$$\left[0,0\right] \tag{B.52}$$

The first-order equation system.

$$\begin{aligned}\Bigg[-\mu\frac{\partial \mathsf{u}^1}{\partial\theta^2}-\mu\frac{\partial \mathsf{u}^2}{\partial\theta^1}=0,\frac{-2\mu\left(-1+\nu\right)}{-1+2\nu}\frac{\partial \mathsf{u}^2}{\partial\theta^2}+\frac{2\mu\nu\frac{\partial \mathsf{u}^1}{\partial\theta^1}}{-1+2\nu}=0,\\ \frac{\partial \mathsf{u}^1}{\partial\theta^1}-\mathsf{u}\hat{}\left[1,1\right]=0,\frac{\partial \mathsf{u}^1}{\partial\theta^2}-\mathsf{u}\hat{}\left[1,2\right]=0,\\ \frac{\partial \mathsf{u}^2}{\partial\theta^1}-\mathsf{u}\hat{}\left[2,1\right]=0,\frac{\partial \mathsf{u}^2}{\partial\theta^2}-\mathsf{u}\hat{}\left[2,2\right]=0\Bigg]\end{aligned} \tag{B.53}$$

APPENDIX C

ORIGINAL APPROACH TO MIXED BVPS

C.1 Overview

This appendix describes the original, “textbook” approach for the solution of two-dimensional sinc boundary value problems originally tried. Although this will work for L_α problems with $\alpha > 1$, it will not work for the present problems. Further, a fully automated implementation of this algorithm is greatly complicated by the presence or absence of certain splines. Lastly, using a second-order system implementation is very restrictive, and yields very large dense matrices.

This short overview is included here to illustrate what *not* to do, and to help readers avoid wasting a large amount of their time.

C.2 Proper approach to mixed boundary value problems

We begin with some known one-dimensional results in section C.2.1; the extension to two dimensions is shown in section C.2.2. The extension to higher dimensions is similar, and will not be shown here.

C.2.1 Sinc methods for one-dimensional problems

In this section, we summarize some available results for sinc interpolation and sinc collocation. The main reference for technical details of this section is the book (Stenger 1993a); most results are proven there.

C.2.1.1 Interpolation and simple collocation

First, some definitions.

The sinc function is defined by

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \tag{C.1}$$

Define the domain D_d by

$$D_d = \{w \in C | \Im(w) < d\} \tag{C.2}$$

Let $\alpha > 0$, and let $L_\alpha(D_d)$ denote the family of functions f with the properties

- f is analytic in D_d;
- for some $c > 0$ and all $z \in D_d$,

$$|f(z)| \leq c \frac{e^{\alpha z}}{(1+|e^z|)^{2\alpha}} \tag{C.3}$$

Now, taking $h = \left(\frac{\pi d}{\alpha N}\right)^{1/2}$, we have the interpolation result

$$f(x) = \sum_{k=-N}^{N} f(kh) \text{sinc} \left(\frac{x - kh}{h}\right) + E(N)$$
$$E(N) = c_1 \sqrt{N} e^{-\sqrt{\pi d \alpha N}} \tag{C.4}$$

for a positive c_1 depending only on f, d and α. Notice that this result is for the real line, and the function must decay at $\pm\infty$.

By first remapping functions approaching a nonzero limit, this can be enhanced to handle nonzero values at $\pm\infty$:

$$f(x) = \sum_{k=-M}^{N} c_k \text{sinc} \left(\frac{x - kh}{h}\right) + c_{N+1} S_\infty(x) + c_{-M-1} S_{-\infty}(x) + E(N) \tag{C.5}$$

$$c_k = f(kh),\ k = -N..N \tag{C.6}$$
$$c_{-M-1} = f(-\infty) \tag{C.7}$$
$$c_{N+1} = f(\infty) \tag{C.8}$$
$$S_{-\infty}(x) = \frac{1}{1 + e^{\alpha x}} \tag{C.9}$$
$$S_\infty(x) = \frac{e^{\alpha x}}{1 + e^{\alpha x}} \tag{C.10}$$

Notice that the summation runs from $-M$ to N; the error bound is of the same form as above.

Defining m as $m = 2N+1$, this means once n digits of accuracy are obtained, one can roughly get $1.4n$ digits by doubling m.

To interpolate a function f defined on $[a,b] \subset \mathbf{R}$ we first make the following definitions:

1. for $[a,b] \in D$, ϕ is a conformal map with $\phi : D \to D_d$
2. $\psi = \phi^{-1}$
3. $\rho = e^{\phi(z)}$
4. ${\gamma_k}^{(h)} = \text{sinc}\left(\frac{\phi(x)-kh}{h}\right)$

Let $\alpha > 0$, and let $L_\alpha(D)$ denote the family of functions F with the properties

- F is analytic in D
- for some $c > 0$ and all $z \in D$,

$$|F(z)| \leq c\frac{\rho(z)^\alpha}{(1+|\rho(z)|)^{2\alpha}} \tag{C.11}$$

Now, taking $h = \left(\dfrac{\pi d}{\alpha N}\right)^{1/2}$, we have the interpolation result

$$f(x) = \sum_{k=-N}^{N} c_k\gamma_k^{(h)}(x) + E(N)$$
$$E(N) = C\sqrt{N}e^{-\sqrt{\pi d\alpha N}} \tag{C.12}$$

for a positive c_1 depending only on f, d and α. Notice that equation C.11 requires f to vanish at the endpoints of the interval. As before, this series can be enhanced to handle nonhomogeneous endpoint values with a simple addition:

$$F(x) = \sum_{k=-M}^{N} c_k\gamma_k^{(h)}(x) + c_{N+1}S_b(x) + c_{-M-1}S_a(x) + E(M,N) \tag{C.13}$$

$$c_k = (F - S_a - S_b)(\psi(kh)), k = -N..N \tag{C.14}$$

$$c_{N+1} = F(b) \tag{C.15}$$

$$c_{-M-1} = F(a) \tag{C.16}$$

and S_a and S_b are cubic splines with value 1 at the left and right endpoints, respectively, and zero derivatives at the endpoints.

As written, the expressions for the c_k are no longer simple function evaluations. This extra work can be shifted to the expansion of F by defining the discrete-orthogonal terms

$$\hat{S}_a(x) = S_a(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.17}$$

$$d_k = S_a(\psi(kh)) \tag{C.18}$$

$$\hat{S}_b(x) = S_b(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.19}$$

$$d_k = S_b(\psi(kh)) \tag{C.20}$$

With these definitions, we have the expansion

$$F(x) = \sum_{k=-N}^{N} c_k \gamma_k^{(h)}(x) + c_b \hat{S}_b(x) + c_a \hat{S}_a(x) \tag{C.21}$$

with $c_k = F(x_k)$, $c_a = F(a)$, $c_b = F(b)$, and $x_k = \psi(kh)$

C.2.1.2 Extensions for mixed boundary value problems

For the solution of mixed boundary value problems, a finite nonzero approximation of the derivative at endpoints is needed. Since the derivatives of the γ_k are unbounded at the endpoint, a nullifier g is introduced to make the derivatives of the series terms also vanish at the endpoints. Then, as before, adding extra terms with the right properties gives a useful basis.

Let $T_a(x)$ and $T_b(x)$ be cubic splines with derivatives of one at the left and right endpoint, respectively, and other values and derivatives zero at the endpoints; then the following series can be used to approximate f on $[a, b]$ when f is specified via a mixed boundary value problem.

$$f(x) = \sum_{k=-M}^{N} c_k \gamma_k(x) + c_b \hat{S}_b(x) + c_a \hat{S}_a(x) + c_{b'} \hat{T}_b(x) + \tag{C.22}$$

$$c_{a'} \hat{T}_a(x) + E(N, M) \tag{C.23}$$

$$\gamma_k = \operatorname{sinc}\left(\frac{\phi(x) - kh}{h}\right) g(x) \tag{C.24}$$

$$x_k = \psi(kh) \tag{C.25}$$

$$g(x) = \frac{1}{\phi'(x)} \tag{C.26}$$

$$c_k = f(x_k)/g(x_k) \tag{C.27}$$

$$c_b = f(b) \tag{C.28}$$

$$c_a = f(a) \tag{C.29}$$

$$c_{b'} = f'(b) \tag{C.30}$$

$$c_{a'} = f'(a) \tag{C.31}$$

$$\hat{S}_a(x) = S_a(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.32}$$

$$d_k = \frac{S_a(x_k)}{g(x_k)} \tag{C.33}$$

$$\hat{S}_b(x) = S_b(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.34}$$

$$d_k = \frac{S_b(x_k)}{g(x_k)} \tag{C.35}$$

$$\hat{T}_a(x) = T_a(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.36}$$

$$d_k = \frac{T_a(x_k)}{g(x_k)} \tag{C.37}$$

$$\hat{T}_b(x) = T_b(x) - \sum_{k=-M}^{N} d_k \gamma_k(x) \tag{C.38}$$

$$d_k = \frac{T_b(x_k)}{g(x_k)} \tag{C.39}$$

All following extensions are based upon this series representation, or subsets of it.

C.2.2 Sinc methods extended for two-dimensional problems

The straightforward way to extend the sinc series to higher dimensions is via tensor products. In this section, two derivations of the series expansion are given and the matrix structure which arises for collocation is described.

In section C.2.2.1, we first show the derivation under the assumption that the unknown can be fully represented by the series (C.22), and using a tensor product to get the two-dimensional extension.

Next, we treat the splines in (C.22) as a change of unknown, so as to produce a homogeneous equation to be satisfied by the simple sum $\sum_{k=-M}^{N} c_k\gamma_k(x)$, and extend this line of thinking to two dimensions – section C.2.2.2.

Extension of the series to multiple domains is considered in section C.2.2.3

C.2.2.1 Tensor product derivation

Let $u(x)$ be represented by the full sum used in equation (C.22), written as a single sum. Thus

$$u(x) = \sum_{k=-M-2}^{N+2} c_k\gamma_k(x) = c_k\gamma_k(x) \tag{C.40}$$

where the last equation is written in summation notation (repeated subscripts are summed over). Then we have

$$\begin{aligned} u(x,y) &= \left[d_{k_1}(y)\right]\gamma_{k_1}(x) \\ &= \left[d_{k_1k_2}\gamma_{k_2}(y)\right]\gamma_{k_1}(x) \\ &= \sum_{k_1=-M_1-2}^{N_1+2}\sum_{k_2=-M_2-2}^{N_2+2} d_{k_1k_2}\gamma_{k_1}(x)\gamma_{k_2}(y) \end{aligned} \tag{C.41}$$

This representation is valid on any rectangular region; regions with other shapes can easily be mapped onto a rectangle. Also, the “rectangle” can be unbounded on one or more sides; only the choice of conformal map (below) changes. Thus, half- or fullspace problems can be solved easily.

Assume for simplicity $N_1 = N_2$, $M_1 = M_2$. Define m as $m = N + M + 1$. Then the series (C.41) has

$$(m+4)^2 = m^2 + 8\,m + 16 \tag{C.42}$$

terms. Notice that this is the most general form possible, used for problems with mixed conditions on all boundaries. Since not every problem has mixed conditions on all boundaries, the nullifier must be selected in conjunction with the splines for each direction and boundary separately before forming the tensor product.

To this end, it is more natural to think of the splines as a remapping of the unknown — the subject of the next section.

For collocation points, starting from $\{x_k | x_k = \psi(kh)\}_{k=-M-1..N+1} \cup \{x_a, x_b\}$ for the one-dimensional case, we get the collocation points as a tensor product also:

$$\{x_{kj} | x_{kj} = (\psi(kh_1), \psi(jh_2))\}_{k=-M_1-1..N_1+1, j=-M_2-1..N_2+1} \cup \tag{C.43}$$

$$\{x_{aj} | x_{aj} = (x_a, \psi(jh_2))\}_{j=-M_2-1..N_2+1} \cup \tag{C.44}$$

$$\{x_{bj} | x_{bj} = (x_b, \psi(jh_2))\}_{j=-M_2-1..N_2+1} \cup \tag{C.45}$$

$$\{x_{ka} | x_{ka} = (\psi(kh_1), y_a)\}_{k=-M_1-1..N_1+1} \cup \tag{C.46}$$

$$\{x_{kb} | x_{kb} = (\psi(kh_1), y_b)\}_{k=-M_1-1..N_1+1} \tag{C.47}$$

Notice that we have $(m+4)^2 = m^2 + 8\,m + 16$ points, as expected.

C.2.2.2 Tensor products revisited

Recall the simple series used for L_α functions in one-dimensional (equation (C.12)). By using a nullifier, this series has zero value and zero derivative at the boundaries. By providing splines for the nonhomogeneous boundary conditions (equation (C.13)), the problem $Lu = f, Bu = g$ is effectively remapped to $Lv = \tilde{f}, Bv = 0$ and this problem can be solved with the sinc-only series (C.12).

The same approach can be taken in two (and higher) dimensions. Starting with the simple sinc series and forming the tensor product, we obtain the representation

$$u(x,y) = \sum_{j=-M_1}^{N_1} \sum_{k=-M_2}^{N_2} c_{jk} \left(S(k,h_1) \circ \phi_1 g_1\right)(x) \left(S(j,h_2) \circ \phi_2 g_2\right)(y) \tag{C.48}$$

valid for $u \in L_\alpha$, $u'(a) = u'(b) = 0$. It thus remains to remap the problem $Lu = f, Bu = g$ to $Lv = \tilde{f}, Bv = 0$.[1] Taking a hint from the previous section, on each boundary of the rectangle, we can use a series of the form

$$\sum_{k=-M_1}^{N_1} \left[\left(S(k,h) \circ \phi_1 g_1\right)(x_1) + \hat{S}_a(x_1) + \hat{S}_b(x_1) + \hat{T}_a(x_1) + \hat{T}_b(x_1)\right] \hat{S}(x_2) \tag{C.49}$$

[1] Note: for single-unknown problems, an explicit remapping can usually be found, so that the resulting problem has only the coefficients of (C.48) as unknowns. However, with multiple unknowns, only the *form* of the remapping is known, and unknown coefficients of this form also go into the matrix system. Thus, this approach is somewhat implicit.

for u's value and tangential derivative, and a series of the form

$$\sum_{k=-M_1}^{N_1} \left[\left(S(k,h) \circ \phi g \right)(x_1) + \hat{S}_a(x_1) + \hat{S}_b(x_1) \right] \hat{T}(x_2) \tag{C.50}$$

for $\dfrac{\partial u}{\partial n}$. The direction of the boundary is x_1, the normal direction x_2. This gives a total of $m^2 + 8m + 24$ terms per unknown. The causes for this larger number are redundancies at the corners of the domain, in the spline-only terms. First, u only has one value at each corner, reducing the total number of terms by 4. Then, the x and y partials at each corner are unique also, further reducing the number of terms by 8. Thus, we are left with $m^2 + 8m + 12$ terms per unknown. The four terms missing here but present in (C.41) are the second-order mixed derivative terms representing $\dfrac{\partial^2}{\partial x \partial y}$ at the corners of the domain.

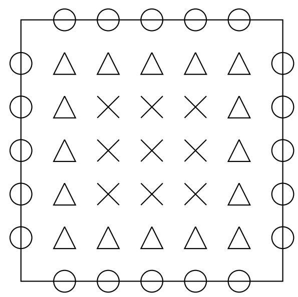
The results in sections C.2.2.1 and C.2.2.2 thus differ at the corners of the domain; numerically, the presence of terms representing the mixed second partial is redundant and these terms are not used.

Next, we derive at the collocation points by extending the one-dimensional problem in a way analogous to the series derivation, rather than by tensor product. First, recall that for one-dimensional problems, three point regions can be distinguished. Always present are interior collocation points, given by $x_k = \psi(kh), k = -M..N$ since these correspond to the sinc part of the series. When the splines representing the value of the unknown at the endpoints of the interval are present ($\hat{S}_{x_a}$ etc.), the collocation point set is expanded to include the boundary points. Lastly, when the splines representing the derivatives at endpoints ($\hat{T}_{x_a}$, etc.) are included, the collocation points are further extended with extra points in the interior: $x_k = \psi(kh), k = \{-M-1, N+1\}$

Selecting the collocation points in two-dimensional analogously we have a point grid as shown in Figure C.1.

Notice that here we have no points at the corners, unlike equations (C.43); this is consistent with the series selection in equations (C.48) through (C.50), since there we also have four terms fewer than equation (C.41).

Figure C.1. The point layout for two-dimensional collocation, for $N = M = 1$. The circles are the boundary points, while the crosses represent the always-present part of the interior points obtained from the one-dimensional tensor product. The points marked with triangles are used only when the derivative spline terms corresponding to them are present. Notice that the points are not really evenly spaced.



C.2.2.3 Multiple-domain problems in two dimensions

Handling of multiple domain problems is straightforward. At the connecting boundaries, the equations involve unknowns from both domains and this simply reflects in the collocation matrix.

Stated another way, the continuous properties of the linear problem are reflected in the discrete approximation via the linear system. Therefore, one has to only consider the matrix implications of domain coupling. These considerations go along with those for general matrix setup.

Further, the method of setting up and solving the system is generalized here. Reference (Morlet, Lybeck, and Bowers 1994) uses the bordering algorithm for a one-dimensional problem on two domains. While it reduces effective matrix sizes for one-dimensional problems, it significantly complicates any algorithm. Also, for two- and higher-dimensional problems, reductions in work are negligible. In reference (Lund and Bowers 1992) and others, equations are manually re-written to reduce nonhomogeneous boundary conditions to homogeneous form. Unfortunately, this approach fails when boundary conditions involve more than one unknown.

REFERENCES

Aho, A. V., R. Sethi, and J. D. Ullman (1986). *Compilers— Principles, Techniques, and Tools*. Reading, MA, USA: Addison-Wesley.

Backus, J. (1978, August). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the Association for Computing Machinery 21*(8), 613–641. Reproduced in "Selected Reprints on Dataflow and Reduction Architectures" ed. S. S. Thakkar, IEEE, 1987, pp. 215-243.

Bogy, D. B. (1968). Edge bonded dissimilar orthogonal elastic wedges under normal and shear loading. *Journal of Applied Mechanics 35*, 460–466.

Chailloux, E., P. Manoury, and B. Pagano (2000, April). *Développement d'applications avec Objective CAML* (1st ed.). O'Reilly & Associates, Inc.

Clements, D. L. (1971). A crack between dissimilar anisotropic media. *Int. J. Eng. Sci. 9*, 257–265.

Demmel, J. W., S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu (1999, July). A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*(3), 720–755. Implementation and documentation available at `http://www.nersc.gov/~xiaoye/SuperLU`.

England, A. H. (1965, June). A crack between dissimilar media. *Journal of Applied Mechanics*, 400–402.

Erdogan, F. (1963, June). Stress distribution in a nonhomogeneous elastic plane with cracks. *Journal of Applied Mechanics*, 232–236.

Folias, E. S. (1965). A finite line crack in a pressurized spherical shell. *International Journal of Fracture*, 20–46.

Folias, E. S. (1975, September). On the three-dimensional theory of cracked plates. *Journal of Applied Mechanics*, 663–674.

Graham, P. (1994). *On Lisp: Advanced Techniques for Common Lisp*. Prentice Hall.

Green, A. E. and W. Zerna (1992). *Theoretical Elasticity*. Dover.

Griffith, A. A. (1924). The theory of rupture. In *Proceedings of the First International Congress of Applied Mechanics*, pp. 55–63.

Hein, V. L. and F. Erdogan (1971). Stress singularities in a two-material wedge. *International Journal of Fracture 7*(3), 317–330.

Houstis, E. N., W. F. Mitchell, and J. R. Rice (1985). Collocation software for second-order elliptic partial differential equations. *ACM Transactions on Mathematical Software 11*(4), 379–412.

Inglis, C. (1913). Stresses in a plate due to the presence of cracks and sharp corners. *Trans. Inst. Naval Architects 55*, 219.

Kaufman, L. (1990, December). TTGU — a package for solving time varying partial differential equations on a union of rectangles. Technical Report 154, AT&T Bell Labs Computing Sciences. Available at `http://cm.bell-labs.com/cm/cs/cstr/154.ps.gz`.

Kernighan, B. W. and R. Pike (1984). *The UNIX Programming Environment*. Upper Saddle River, NJ 07458, USA: Prentice-Hall.

Kowalski, M. A., K. A. Sikorski, and F. Stenger (1995). *Selected Topics in Approximation and Computation*. Oxford University Press.

Leroy, X. (1995). Le système caml special light: modules et compilation efficace en caml. Technical report, INRIA. This report describes the precursor to OCAML. Available at `http://pauillac.inria.fr/~xleroy/publi/caml-special-light-rr.ps.gz`.

Leroy, X. (1997). Objective caml. Online, `http://pauillac.inria.fr/ocaml`. Most recent version at `ftp://ftp.inria.fr/lang/caml-light/ocaml-3.00.tar.gz`.

Levine, J. R., T. Mason, and D. Brown (1992). *`lex` & `yacc`* (Second ed.). 981 Chestnut Street, Newton, MA 02164, USA: O'Reilly & Associates, Inc.

Li, L. and S. Nemat-Nasser (1990). Interface cracks in anisotropic dissimilar materials: an analytic solution. *J. Mech. Phys. Solids 39*, 113–144.

Lund, J. and K. L. Bowers (1992). *Sinc Methods for Quadrature and Differential Equations*. SIAM.

Lybeck, N. J. and K. L. Bowers (1994). The sinc-Galerkin patching method for Poisson's equation. *Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics 1*, 325–328. Also available at `http://www.math.montana.edu/~bowers/publications/imacs94.ps`.

Lybeck, N. J. and K. L. Bowers (1996). Sinc methods for domain decomposition. *Applied Mathematics and Computation 75*, 13–41. Also available at `http://www.math.montana.edu/~bowers/publications/dd1.ps`.

Morlet, A. et al. (1997). The schwarz alternating sinc domain decomposition method. *Applied Numerical Math. 25*, 461–483.

Morlet, A. C., N. J. Lybeck, and K. L. Bowers (1994, October). Sinc domain decomposition methods I: The direct approach. The contents of this paper are similar to (Lybeck and Bowers 1996).

Morlet, A. C., N. J. Lybeck, and K. L. Bowers (1999). Convergence of the sinc overlapping domain decomposition method. *Applied Mathematics and Computation 98*, 209–227. Also available at `http://www.math.montana.edu/~bowers/publications/overconv.ps`.

Penado, F. E. and E. S. Folias (1989). The three dimensional stress field around a cylindrical inclusion in a plate of arbitrary thickness. *International Journal of Fracture 39*, 129–145.

Raymond, E. (1991). *The New Hacker's Dictionary*. Cambridge, MA, USA: MIT Press. This book corresponds to version 2.9.6 of the on-line jargon file. The latest (at the time of writing) is version 2.9.12 (`jargon2912.txt.z`) which is available by anonymous `ftp` from `prep.ai.mit.edu` (in `/pub/gnu`) or `wuarchive.wustl.edu` (in `mirrors/gnu`). Changes since the publication of this book can be found in the file `jargon-upd.z`. (`*.z` are files compressed by GNU `zip` (`gzip`)).

Rice, J. R. and G. C. Sih (1965). Plane problems of cracks in dissimilar media. *ASME J. Appl. Mech. 32*, 418 – 423.

Schwing, J. (1976). *Numerical Solution of Integral Equations in Potential Theory Problems*. Ph. D. thesis, University of Utah.

Sneddon, I. N. (1946). The distribution of stress in the neighbourhood of a crack in an elastic solid. *Proc. Roy. Soc. London A187*, 229–260.

Sneddon, I. N. (1995). *Fourier Transforms*. Dover.

Stenger, F. (1993a). *Numerical Methods Based on Sinc and Analytic Functions*. Springer-Verlag.

Stenger, F. (1993b). Sincpack. Available at `http://www.cs.utah.edu/~stenger/PACKAGES/SincPack.tar`.

Stenger, F. (1993c). Summary of sinc approximation. Available at `http://www.cs.utah.edu/~stenger/PACKAGES/SincPack.ps`.

Zhong, F. H. and E. S. Folias (1992). The 3d stress field of a fiber embedded into a matrix and subjected to an axial load. *Computational Mechanics 9*, 1–15.