

# Documentation for Collocation Software <sup>1</sup>

Stephen H. Brill

<sup>1</sup>RCGRD Publication #98-01, Research Center for Groundwater Remediation Design, University of Vermont, Burlington, Vermont, 1998

## **Abstract**

This document describes `coll1d.c` and `coll2d.c`, computer programs written in the C programming language, that solve differential equations (DEs) discretized by Hermite collocation. Chapter 1 provides the details for `coll1d.c`, which is for DEs in one spatial dimension. The program `coll2d.c`, which solves DEs in two spatial dimensions, is discussed in Chapter 2.

# Contents

<b>List of Figures</b>	<b>3</b>
<b>List of Tables</b>	<b>4</b>
<b>1 Problems in One Spatial Dimension</b>	<b>5</b>
1.1 Introduction . . . . .	5
1.2 Solution of ODEs . . . . .	6
1.2.1 The Hermite interpolating polynomial . . . . .	6
1.2.2 Collocation discretization . . . . .	7
1.3 Solution of PDEs . . . . .	9
1.4 Discussion of <code>coll1d.c</code> code . . . . .	12
1.4.1 The file <code>struct1d.h</code> . . . . .	12
1.4.2 The file <code>coll1d.c</code> . . . . .	13
1.4.3 The file <code>fun1d.h</code> . . . . .	17
<b>2 Problems in Two Spatial Dimensions</b>	<b>18</b>
2.1 Introduction . . . . .	18
2.2 Collocation in two dimensions . . . . .	19
2.2.1 Hermite interpolating polynomial . . . . .	19
2.2.2 Collocation points . . . . .	19
2.2.3 Boundary conditions . . . . .	21
2.3 Numbering of equations and unknowns . . . . .	21
2.3.1 The preliminary ordering scheme . . . . .	22
2.3.2 The red-black ordering scheme . . . . .	24
2.4 Preconditioned conjugate gradient methods . . . . .	26
2.5 The time-dependent PDE (2.1) . . . . .	28
2.6 Discussion of <code>coll2d.c</code> code . . . . .	28
2.6.1 The file <code>struct2d.h</code> . . . . .	28

2.6.2	The file <code>coll2d.c</code> . . . . .	30
2.6.3	The file <code>fun2d_p1.h</code> . . . . .	33
2.6.4	The file <code>fun2d_p2.h</code> . . . . .	34
2.6.5	The file <code>fun2d_p3.h</code> . . . . .	35
<b>Bibliography</b>		<b>36</b>
<b>A Source Code for Collocation Software</b>		<b>38</b>

# List of Figures

1.1	Functions $f_L$ , $f_R$ , $g_L$ , and $g_R$ . . . . .	7
2.1	Location of degrees of freedom for two-dimensional Hermite interpolation . . . . .	20
2.2	Preliminary numbering of equations and unknowns for two-dimensional collocation . . . . .	22
2.3	Structure of matrix $\mathbf{M}$ arising from preliminary numbering scheme . . . . .	23
2.4	Red-Black numbering of equations and unknowns for two-dimensional collocation . . . . .	25
2.5	Matrix structure arising from Red-Black numbering of equations and unknowns . . . . .	26

# List of Tables

2.1	Coding for corner node unknowns . . . . .	33
A.1	Files for collocation software . . . . .	38

# Chapter 1

## Problems in One Spatial Dimension

### 1.1 Introduction

This chapter describes `coll1d.c`, a computer program that numerically solves, via Hermite collocation, the partial differential equation (PDE)

$$Q(x) \frac{\partial u}{\partial t} + A(x, t) \frac{\partial^2 u}{\partial x^2} + B(x, t) \frac{\partial u}{\partial x} + C(x, t) u = H(x, t) \quad (1.1)$$

with appropriate initial and boundary conditions. If  $Q(x) \equiv 0$  and the functions  $A$ ,  $B$ ,  $C$ , and  $H$  all lack time dependence, then `coll1d.c` solves the ordinary differential equation (ODE)

$$A(x) \frac{d^2 u}{dx^2} + B(x) \frac{du}{dx} + C(x) u = H(x) \quad (1.2)$$

with appropriate boundary conditions.

This chapter is organized as follows. We first discuss the Hermite interpolating polynomial and then the collocation discretization of (1.2). We next introduce the finite difference approximation of the temporal derivative in (1.1) and the resulting collocation discretization of (1.1). Finally, we describe particulars of the computer code.

## 1.2 Solution of ODEs

### 1.2.1 The Hermite interpolating polynomial

Let  $u(x)$  be a differentiable function defined on the interval  $\mathcal{I} = [a, b]$ . Partition  $\mathcal{I}$  into  $m$  linear finite elements  $[x_j, x_{j+1}]$ ,  $j = 0, 1, 2, \dots, m-1$ , where the  $m+1$  mesh points (or nodes) are  $a = x_0 < x_1 < x_2 < \dots < x_m = b$ . Let

$$h_j = x_{j+1} - x_j, \quad (1.3)$$

$j = 0, 1, 2, \dots, m-1$ .

Consider the Hermite cubic polynomials, defined for  $\eta$  in the interval  $[-\frac{1}{2}, \frac{1}{2}]$ :

$$f_j(x) = \begin{cases} f_L(\eta) = \frac{1}{2}(1+2\eta)^2(1-\eta), & x_{j-1} \leq x = x_j + \left(\eta - \frac{1}{2}\right)h_{j-1} \leq x_j \\ f_R(\eta) = \frac{1}{2}(1-2\eta)^2(1+\eta), & x_j \leq x = x_j + \left(\eta + \frac{1}{2}\right)h_j \leq x_{j+1} \\ 0, & \text{otherwise} \end{cases} \quad (1.4)$$

and

$$g_j(x) = \begin{cases} g_L(\eta, h_{j-1}) = \frac{h_{j-1}}{8}(2\eta+1)^2(2\eta-1), & x_{j-1} \leq x = x_j + \left(\eta - \frac{1}{2}\right)h_{j-1} \leq x_j \\ g_R(\eta, h_j) = \frac{h_j}{8}(2\eta-1)^2(2\eta+1), & x_j \leq x = x_j + \left(\eta + \frac{1}{2}\right)h_j \leq x_{j+1} \\ 0, & \text{otherwise} \end{cases}, \quad (1.5)$$

$j = 0, 1, 2, \dots, m$ . The functions  $f_L$ ,  $f_R$ ,  $g_L$ , and  $g_R$  are depicted in Figure 1.1. The functions  $f_j(x)$  and  $g_j(x)$  are defined such that

$$f_j(x_i) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} \end{cases}$$

and

$$\frac{dg_j}{dx}(x_i) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} \end{cases}$$

for  $i, j = 0, 1, 2, \dots, m$ .

If we know the values  $u_j = u(x_j)$  and  $u'_j = \frac{du}{dx}(x_j)$  for  $j = 0, 1, 2, \dots, m$ , then the piecewise cubic polynomial interpolating the values  $u_j$  and  $u'_j$ ,  $j = 0, 1, 2, \dots, m$  is

$$\hat{u}(x) = \sum_{j=0}^m [u_j f_j(x) + u'_j g_j(x)]. \quad (1.6)$$

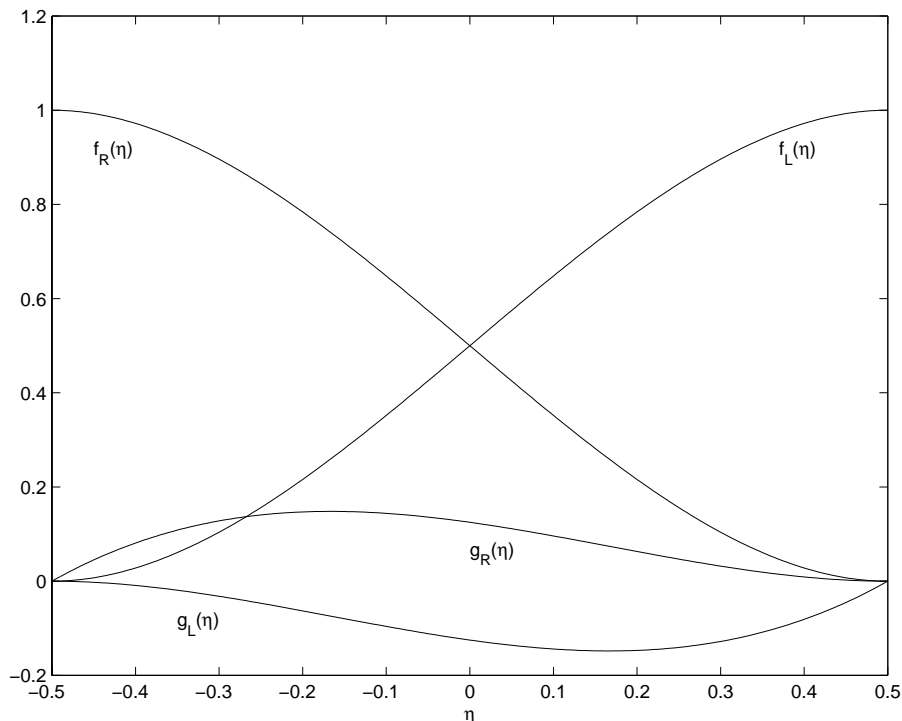


Figure 1.1: Functions  $f_L$ ,  $f_R$ ,  $g_L$ , and  $g_R$

### 1.2.2 Collocation discretization

Recall the ODE (1.2)

$$\mathcal{L}[u](x) = A(x) \frac{d^2 u}{dx^2} + B(x) \frac{du}{dx} + C(x) u = H(x). \quad (1.7)$$

If we introduce the interpolating polynomial (1.6) into the differential equation (1.7), we obtain

$$\mathcal{L}[\hat{u}](x) - H(x) = E(x), \quad (1.8)$$

where  $E(x)$  is an error function. Our goal is to determine  $u_j$  and  $u'_j$  for  $j = 0, 1, 2, \dots, m$  while controlling  $E(x)$ .

The collocation discretization proceeds by selecting two distinct points in the interior of each of the  $m$  finite elements and enforcing the condition  $E(x) = 0$  at these  $2m$  “collocation points”. If we let  $c_0 < c_1 < c_2 < \dots <$



$$\mathbf{x} = \left[ u'_0 \quad u_1 \mid u'_1 \quad u_2 \mid u'_2 \quad u_3 \right]^T,$$

and

$$\mathbf{b} = \begin{bmatrix} H(c_0) - \mathcal{L}[f_0](c_0) u_a \\ H(c_1) - \mathcal{L}[f_0](c_1) u_a \\ \hline H(c_2) \\ H(c_3) \\ \hline H(c_4) - \mathcal{L}[g_3](c_4) u'_b \\ H(c_5) - \mathcal{L}[g_3](c_5) u'_b \end{bmatrix}.$$

Because  $\mathbf{M}$  has a  $2 \times 2$  block tridiagonal structure, efficient code may be written for the solution of (1.12), the collocation discretization of the ODE (1.2).

### 1.3 Solution of PDEs

We now apply the ideas introduced in the previous section to the solution of the PDE (1.1)

$$Q(x) \frac{\partial u}{\partial t} + A(x, t) \frac{\partial^2 u}{\partial x^2} + B(x, t) \frac{\partial u}{\partial x} + C(x, t) u = H(x, t). \quad (1.13)$$

By analogy to (1.7), let

$$\mathcal{L}[u](x, t) = A(x, t) \frac{\partial^2 u}{\partial x^2} + B(x, t) \frac{\partial u}{\partial x} + C(x, t) u \quad (1.14)$$

We discretize the time derivative in (1.13) via a first-order Taylor series approximation:

$$\frac{\partial u}{\partial t} \approx \frac{u^{(n+1)} - u^{(n)}}{\Delta t}, \quad (1.15)$$

where the symbol  $u^{(n)}$  denotes the value of  $u$  at time level  $n$ . By analogy to (1.14), define

$$\mathcal{J}[u](x, t) = Q(x) u. \quad (1.16)$$

Introducing (1.14), (1.15) and (1.16) into (1.13) yields

$$\frac{\mathcal{J}[u](x, t_{n+1}) - \mathcal{J}[u](x, t_n)}{\Delta t} + \mathcal{L}[u](x, t) = H(x, t). \quad (1.17)$$



and

$$\frac{du}{dx}(b, t_{n+1}) = u_m^{(n+1)} = u_b^{(n+1)},$$

we obtain the equation analogous to (1.12):

$$\left(\mathbf{T} + \tau\mathbf{M}^{(n+1)}\right) \mathbf{x}^{(n+1)} = \left(\tilde{\mathbf{T}} - \tau\tilde{\mathbf{M}}^{(n)}\right) \tilde{\mathbf{x}}^{(n)} + \tau\tilde{\mathbf{b}}^{(n)} + \mathbf{c}^{(n+1)}, \quad (1.21)$$

where (for the case  $m = 3$ )

$$\mathbf{M}^{(k)} = \left[ \begin{array}{cc|cc|cc} \mathcal{L}[g_0](c_0, t_k) & \mathcal{L}[f_1](c_0, t_k) & \mathcal{L}[g_1](c_0, t_k) & & & \\ \mathcal{L}[g_0](c_1, t_k) & \mathcal{L}[f_1](c_1, t_k) & \mathcal{L}[g_1](c_1, t_k) & & & \\ \hline & \mathcal{L}[f_1](c_2, t_k) & \mathcal{L}[g_1](c_2, t_k) & \mathcal{L}[f_2](c_2, t_k) & \mathcal{L}[g_2](c_2, t_k) & \\ & \mathcal{L}[f_1](c_3, t_k) & \mathcal{L}[g_1](c_3, t_k) & \mathcal{L}[f_2](c_3, t_k) & \mathcal{L}[g_2](c_3, t_k) & \\ \hline & & & \mathcal{L}[f_2](c_4, t_k) & \mathcal{L}[g_2](c_4, t_k) & \mathcal{L}[f_3](c_4, t_k) \\ & & & \mathcal{L}[f_2](c_5, t_k) & \mathcal{L}[g_2](c_5, t_k) & \mathcal{L}[f_3](c_5, t_k) \end{array} \right],$$

$$\mathbf{T} = \left[ \begin{array}{cc|cc|cc} \mathcal{J}[g_0](c_0) & \mathcal{J}[f_1](c_0) & \mathcal{J}[g_1](c_0) & & & \\ \mathcal{J}[g_0](c_1) & \mathcal{J}[f_1](c_1) & \mathcal{J}[g_1](c_1) & & & \\ \hline & \mathcal{J}[f_1](c_2) & \mathcal{J}[g_1](c_2) & \mathcal{J}[f_2](c_2) & \mathcal{J}[g_2](c_2) & \\ & \mathcal{J}[f_1](c_3) & \mathcal{J}[g_1](c_3) & \mathcal{J}[f_2](c_3) & \mathcal{J}[g_2](c_3) & \\ \hline & & & \mathcal{J}[f_2](c_4) & \mathcal{J}[g_2](c_4) & \mathcal{J}[f_3](c_4) \\ & & & \mathcal{J}[f_2](c_5) & \mathcal{J}[g_2](c_5) & \mathcal{J}[f_3](c_5) \end{array} \right],$$

$$\mathbf{x}^{(k)} = \left[ u_0^{(k)} \quad u_1^{(k)} \mid u_1^{(k)} \quad u_2^{(k)} \mid u_2^{(k)} \quad u_3^{(k)} \right]^T,$$

and

$$\mathbf{c}^{(n+1)} = \left[ \begin{array}{c} \tau H(c_0, t_{n+1}) - (\{\mathcal{J} + \tau\mathcal{L}\}[f_0](c_0, t_{n+1})) u_a^{(n+1)} \\ \tau H(c_1, t_{n+1}) - (\{\mathcal{J} + \tau\mathcal{L}\}[f_0](c_1, t_{n+1})) u_a^{(n+1)} \\ \hline \tau H(c_2, t_{n+1}) \\ \tau H(c_3, t_{n+1}) \\ \hline \tau H(c_4, t_{n+1}) - (\{\mathcal{J} + \tau\mathcal{L}\}[g_3](c_4, t_{n+1})) u_b^{(n+1)} \\ \tau H(c_5, t_{n+1}) - (\{\mathcal{J} + \tau\mathcal{L}\}[g_3](c_5, t_{n+1})) u_b^{(n+1)} \end{array} \right]. \quad (1.22)$$

Finally, we note that it is implicit in (1.21) that the type of boundary conditions may change from one time level to the next. That is, with respect to  $\mathbf{c}^{(n+1)}$  (see (1.22)), a Dirichlet boundary condition was applied at  $a$  while a Neumann boundary condition was applied at  $b$  for the computation of  $\mathbf{x}^{(n+1)}$ . However, it could be the case that when computing  $\mathbf{x}^{(n)}$  (i.e., at the previous time step) we used, say, a Neumann boundary condition at  $a$  and a Dirichlet boundary condition at  $b$ . We note this because the `coll1d.c` software does not allow this generality of allowing the type of boundary condition to vary

from time step to time step. Thus, with respect to (1.21), we write the time-stepping scheme that `coll1d.c` uses to solve (1.13) as

$$\left(\mathbf{T} + \tau \mathbf{M}^{(n+1)}\right) \mathbf{x}^{(n+1)} = \left(\mathbf{T} - \bar{\tau} \mathbf{M}^{(n)}\right) \mathbf{x}^{(n)} + \bar{\mathbf{c}}^{(n)} + \mathbf{c}^{(n+1)}, \quad (1.23)$$

where

$$\bar{\mathbf{c}}^{(n)} = \begin{bmatrix} \bar{\tau} H(c_0, t_n) + (\{\mathcal{J} - \bar{\tau} \mathcal{L}\} [f_0](c_0, t_n)) u_a^{(n)} \\ \bar{\tau} H(c_1, t_n) + (\{\mathcal{J} - \bar{\tau} \mathcal{L}\} [f_0](c_1, t_n)) u_a^{(n)} \\ \hline \bar{\tau} H(c_2, t_n) \\ \bar{\tau} H(c_3, t_n) \\ \hline \bar{\tau} H(c_4, t_n) + (\{\mathcal{J} - \bar{\tau} \mathcal{L}\} [g_3](c_4, t_n)) u_b^{(n)} \\ \bar{\tau} H(c_5, t_n) + (\{\mathcal{J} - \bar{\tau} \mathcal{L}\} [g_3](c_5, t_n)) u_b^{(n)} \end{bmatrix}. \quad (1.24)$$

## 1.4 Discussion of `coll1d.c` code

Assuming that the derivation of (1.23) is well understood, it should be fairly easy to comprehend the computer program `coll1d.c`. We will describe each section of the code and how it relates to the discussion above. One should note that the `structs` of `coll1d.c` are defined in the file `struct1d.h` and that `coll1d.c` calls subroutines from the file `fun1d.h`. Complete source code for all three files is included in the Appendix.

### 1.4.1 The file `struct1d.h`

This file contains C language `structs` that `coll1d.c` utilizes. The motivation for the creation of these `structs` lies in the observation that the matrix  $\mathbf{T} + \tau \mathbf{M}^{(n+1)}$  in (1.23) is  $2 \times 2$  block tridiagonal and that code should therefore be written which exploits this structure.

The first `struct` defined in the file `struct1d.h` is `struct v2`, which can be thought of as a column vector with two entries. The last two `structs`, namely `struct kvec2` and `struct gvec2`, should be considered to be composed of, respectively,  $m$  and  $m - 1$  copies of `struct v2`. Thus a `struct kvec2` is a column vector of  $2m$  entries while a `struct gvec2` is a column vector of  $2(m - 1)$  entries.

A `struct m22` should be considered to be a  $2 \times 2$  matrix.

The most complicated of the `structs` is `struct triblock22`, which represents the storage required to hold the non-zero entries of matrix  $\mathbf{T} + \tau \mathbf{M}^{(n+1)}$ .

A `struct triblock22` is composed of two sets (denoted by `c` and `b`) of  $m-1$  `struct v2s` each and one set (denoted by `a`) of  $m$  `struct m22s`.

If, for example, we wish to solve the matrix equation

$$\mathbf{G}\mathbf{x} = \mathbf{b} \tag{1.25}$$

for the case  $m = 3$ , we write

$$\left[ \begin{array}{cc|cc} G.a.0.00 & G.a.0.01 & G.b.0.0 & \\ G.a.0.10 & G.a.0.11 & G.b.0.1 & \\ \hline & G.c.0.0 & G.a.1.00 & G.a.1.01 & G.b.1.0 \\ & G.c.0.1 & G.a.1.10 & G.a.1.11 & G.b.1.1 \\ \hline & & G.c.1.0 & G.a.2.00 & G.a.2.01 \\ & & G.c.1.0 & G.a.2.10 & G.a.2.11 \end{array} \right] \begin{bmatrix} x.0.0 \\ x.0.1 \\ x.1.0 \\ x.1.1 \\ x.2.0 \\ x.2.1 \end{bmatrix} = \begin{bmatrix} b.0.0 \\ b.0.1 \\ b.1.0 \\ b.1.1 \\ b.2.0 \\ b.2.1 \end{bmatrix}. \tag{1.26}$$

The numbering system given in (1.26) corresponds to the non-zero entries of matrix  $\mathbf{G}$  being stored in a `struct triblock22`, while the entries of vectors  $\mathbf{x}$  and  $\mathbf{b}$  are each stored in a `struct kvec2`. (It may be helpful to keep in mind that, in the C language, a list of  $n$  elements is generally numbered  $0, 1, 2, \dots, n-1$ .)

## 1.4.2 The file `coll1d.c`

This file contains the commands that solve the PDE (1.1) or the ODE (1.2). It also calls subroutines contained in the file `fun1d.h`. We will proceed through the file `coll1d.c`, discussing salient features of the software.

### Parameters for problem definition

For those who want to use `coll1d.c` as a “black box” solver, we will first describe those parts of the code that define a specific problem. They are:

- `#define m` (line 1). This line defines the number of finite elements  $m$  in the problem. Its value is presently set to  $m = 4$ .
- `#define sstol` (line 2). When solving the PDE (1.1), `coll1d.c` will stop its time-stepping scheme once the solution has reached a steady state, defined as

$$\|\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}\|_{\infty} < \text{sstol},$$

where

$$\mathbf{u}^{(k)} = \left[ u_0^{(k)} \quad u_1^{(k)} \quad u_2^{(k)} \quad \cdots \quad u_m^{(k)} \right]^T. \tag{1.27}$$

The value of `sstol` is presently set to 0.001.

- `#define A(x,t), B(x,t), C(x,t), and Q(x)` (lines 8-11). These functions are found in the PDE (1.1). If solving the ODE (1.2), we require  $Q(x) \equiv 0$  and the functions  $A$ ,  $B$ , and  $C$  to be independent of  $t$ .

- `#define S(x,t) and dSx(x,t)` (lines 16-17). The solution of the PDE (1.1) requires appropriate initial conditions for the functions  $u(x, t)$  and  $\frac{\partial u}{\partial x}(x, t)$ . Use the function  $S(x, t)$  to define initial values for  $u(x, t)$  and  $dSx(x, t)$  to define initial values for  $\frac{\partial u}{\partial x}(x, t)$  using a fixed initial value of  $t = \text{init\_t}$  (see below).

- `#define H(x,t)` (line 24). This defines the forcing function  $H(x, t)$  for the PDE (1.1). If solving the ODE (1.2), we require the function  $H$  to be independent of  $t$ . Its value is presently defined in terms of a known analytical solution.

- `#define BCTypeL, BCTypeR` (lines 31, 33). The code handles two types of boundary conditions. A Type 1 (or Dirichlet) boundary condition specifies  $u$  at a boundary node while a Type 2 (or Neumann) boundary condition specifies  $\frac{\partial u}{\partial x}$  at a boundary node. The letters `L` and `R` after `BCType` represent the left endpoint (i.e.  $a$ ) and the right endpoint (i.e.  $b$ ), respectively. Thus we presently have a Neumann boundary condition specified at  $a$  and a Dirichlet boundary condition specified at  $b$ .

- `#define BCValuL(t), BCValuR(t)` (lines 32, 34). These functions specify the value assigned to the boundary conditions at  $a$  and  $b$ , respectively. Their values are presently defined in terms of a known analytical solution. If solving the PDE (1.1), these functions may be defined to depend on  $t$ . If solving the ODE (1.2), these functions must be constant scalars.

- `/* define nodes */` (lines 81-85). Here we define the nodes  $a = x_0 < x_1 < x_2 < \dots < x_m = b$ . The number of nodes defined must be  $m + 1$ .

- `/* define xi values */` (lines 90-91). To achieve maximum accuracy (assuming certain smoothness conditions), we select the location of the collocation points defined by  $\eta$  in (1.4) and (1.5) to be  $\eta = \pm\xi$ , where  $\xi = \frac{1}{\sqrt{12}}$  (i.e., the collocation points coincide with the points of Gaussian quadrature) [3], [4], [7]. However, the parameter  $\xi$  may assume any value on the interval  $(0, \frac{1}{2})$ . The code allows  $\xi$  to assume a different value on each finite element. Presently, however,  $\xi$  is set to the Gaussian value of  $\frac{1}{\sqrt{12}}$  for all finite elements.

- `/* time control */` (lines 96-99). The parameter `init_t` is the value of  $t$  at which the initial values are defined. The parameter `dt` is the time

step  $\Delta t$ , which does not change in value from time step to time step. The parameter `tsteps` gives the maximum number of time steps over which the code runs. Its value should be set to 1 if solving the ODE (1.2). The value of weighting parameter  $\theta$  in (1.18) is given by `theta`.

## Description of the code

We discuss in this section the code for `coll1d.c`, some of which is discussed above. We will proceed from beginning to end while not repeating ourselves if we covered a certain topic above.

- `/* exact solution */` (lines 16-19). When developing code, it is often helpful to test it on a problem for which the analytical solution is known. These lines allow the user to define the analytical solution  $u(x, t) = \mathbf{S}(\mathbf{x}, \mathbf{t})$  and its derivative  $\frac{\partial u}{\partial x}(x, t) = \mathbf{dSx}(\mathbf{x}, \mathbf{t})$ . Using these functions permit easy definition of the boundary conditions assuming that the analytical solution is known. Using also the definitions  $\frac{\partial^2 u}{\partial x^2}(x, t) = \mathbf{d2Sx}(\mathbf{x}, \mathbf{t})$  and  $\frac{\partial u}{\partial t}(x, t) = \mathbf{dSt}(\mathbf{x}, \mathbf{t})$  permit easy definition of the forcing function assuming, again, that the analytical solution is known. As discussed above, we define the initial conditions using the functions  $\mathbf{S}(\mathbf{x}, \mathbf{t})$  and  $\mathbf{dSx}(\mathbf{x}, \mathbf{t})$  with a fixed initial value of `t` set to `init_t`.

- `#include...` (lines 36-38). These are C library files that are required.

- `#define hermite...` (lines 40-53). Here we define the Hermite polynomials  $f_L$ ,  $f_R$ ,  $g_L$ , and  $g_R$  (see (1.4) and (1.5)) and their first and second derivatives with respect to  $x$ . Note that these functions are defined in terms of the *local* parameter  $\eta$  (and not in terms of the *global* parameter  $x$ ).

- `#define L...` (lines 55-58). Here we apply the differential operator  $\mathcal{L}$  (see (1.7) and (1.14)) to the Hermite polynomials  $f_L$ ,  $f_R$ ,  $g_L$ , and  $g_R$ .

- `#define J...` (lines 60-63). Here we apply the differential operator  $\mathcal{J}$  (see (1.16)) to the Hermite polynomials  $f_L$ ,  $f_R$ ,  $g_L$ , and  $g_R$ .

- (lines 70-76) declaration of variables.

- `/* define finite element... */` (lines 106-117). Here we define the finite element widths  $h_j$  (see (1.3)). We also define the locations of the collocation points, stored in two different ways. The vector `c` stores the collocation point locations in *local*  $\eta$  coordinates (as required for the Hermite functions). The vector `cc` stores the collocation point locations in *global*  $x$  coordinates (as required for the forcing function  $H$ ).

- `/* initial conditions */` (lines 122-140). Here the `struct kvec2` `u` is assigned the values for  $\mathbf{x}^{(0)}$  in (1.23). Note that the first two and last

two components of  $\mathbf{x}^{(0)}$  depend on the type of boundary condition specified at the endpoints.

- `/* time-independent... */` (lines 145-176). The vector `BCTL` (respectively, `BCTR`) stores the values of the differential operator  $\mathcal{J}$  applied to the appropriate Hermite polynomials evaluated at the collocation points in the left- (respectively, right-)most finite element as required for the computation of (1.22) and (1.24). For example, with reference to (1.22) and (1.24), `BCTL` stores  $\mathcal{J}[f_0](c_0)$  and  $\mathcal{J}[f_0](c_1)$  while `BCTR` stores  $\mathcal{J}[g_3](c_4)$  and  $\mathcal{J}[g_3](c_5)$ . The vectors `BCML` and `BCMR` play an analogous role with the differential operator  $\mathcal{L}$ . That is, again with reference to (1.22) and (1.24), `BCML` stores  $\mathcal{L}[f_0](c_0, t_0)$  and  $\mathcal{L}[f_0](c_1, t_0)$  while `BCMR` stores  $\mathcal{L}[g_3](c_4, t_0)$  and  $\mathcal{L}[g_3](c_5, t_0)$ .

- `/* matrices... */` (lines 181-182). Here we define the matrices  $\mathbf{T}$  and  $\mathbf{M}^{(0)}$  in (1.23).

- `/* initial RHS... */` (lines 187-193). Here we define the vector  $\tilde{\mathbf{b}}^{(0)}$  in (1.20).

We now begin the main loop, where each iteration is one simulated time step.

- `/* matrix * solution... */` (lines 203-204). Here we compute the vector  $(\mathbf{T} - \tau\mathbf{M}^{(n)})\mathbf{x}^{(n)}$  in (1.23).

- `/* forcing function... */` (lines 209-217). Here we define `cbar` (see (1.24)).

- `/* Make matrix G */` (lines 227-228). Here we define the matrix  $\mathbf{G} = \mathbf{T} + \tau\mathbf{M}^{(n+1)}$  (see (1.23)).

- `/* BCs at new time step... */` (lines 233-253). Here we compute then values for `BCML` and `BCMR` (see discussion re: lines 145-176) for the new time level  $n + 1$ .

- `/* evaluate RHS... */` (lines 258-271). Here we define `cvec` (see (1.22)).

- `/* form final RHS... */` (lines 276-277). Here we compute `bcombo`  $= (\mathbf{T} - \tau\mathbf{M}^{(n)})\mathbf{x}^{(n)} + \bar{\mathbf{c}}^{(n)} + \mathbf{c}^{(n+1)}$  (see (1.23)).

- `/* solve... */` (line 282). We compute the solution  $\mathbf{x}^{(n+1)}$  (stored in `struct kvec2 u`) of (1.23). See below for more detail.

- `/* compute results */` (lines 287-320). The vector  $\mathbf{U} = \mathbf{u}^{(n)}$  (see (1.27)) is copied into `oldU`. Then the entries of vector  $\mathbf{x}^{(n+1)}$  and the boundary conditions at  $t = t_{n+1}$  are incorporated into the vectors  $\mathbf{U}$  (for values of  $u(x, t_{n+1})$ ) and  $\mathbf{dU}$  (for values of  $\frac{\partial u}{\partial x}(x, t_{n+1})$ ).

- `subvec...` (line 322). Subtracts `oldU` from `U` for steady state check.
- `/* results to screen */` (lines 327-331). Prints to the screen the time step number  $n + 1$ , the value of  $t_{n+1}$ , and the value of  $\Delta t = t_{n+1} - t_n$ . It then prints a table, the columns of which are  $x_j$ ,  $u_j$ , and  $u'_j$  for  $j = 0, 1, \dots, m$ . Presently commented out between the `/*` and `*/` (lines 333-337) is code to print the values of the analytical solution (if available) at  $t = t_{n+1}$ .
- (lines 339-345). Here the steady state check is performed. The value of  $\|\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}\|_\infty$  is printed to the screen. Assuming steady state has not been reached and that more time steps are permitted, a new iteration (i.e., time step) now begins.

### 1.4.3 The file `fun1d.h`

The most important subroutine in this file is `tridiag2` (lines 218-256), which solves the matrix equation (1.25), where the numbering scheme for  $\mathbf{G}$ ,  $\mathbf{x}$ , and  $\mathbf{b}$  is given in (1.26). We note that  $\mathbf{x}$  and  $\mathbf{b}$  are partitioned into  $m$  vectors of length two while  $\mathbf{G}$  is partitioned into an  $m \times m$  tridiagonal array of  $2 \times 2$  matrices. Furthermore, while the main diagonal (`a`) of  $\mathbf{G}$  contains dense  $2 \times 2$  matrices, the superdiagonal (`b`) of  $\mathbf{G}$  contains  $2 \times 2$  matrices whose second column contains all zeros and the subdiagonal (`c`) of  $\mathbf{G}$  contains  $2 \times 2$  matrices whose first column contains all zeros. We are therefore motivated to write code that exploits the special sparsity of this structure.

The routine `tridiag2`, based upon the algorithm given in [9], uses the common forward elimination — back substitution technique. To avoid performing unnecessary calculations that require, say, adding or multiplying with a value of zero, many small routines (e.g., `cmultb` (lines 65-70)) are necessary.

# Chapter 2

## Problems in Two Spatial Dimensions

### 2.1 Introduction

This chapter describes `coll2d.c`, a computer program that numerically solves, via Hermite collocation, the PDE

$$Q(x, y) \frac{\partial u}{\partial t} + A(x, y, t) \frac{\partial^2 u}{\partial x^2} + B(x, y, t) \frac{\partial^2 u}{\partial x \partial y} + C(x, y, t) \frac{\partial^2 u}{\partial y^2} + D(x, y, t) \frac{\partial u}{\partial x} + E(x, y, t) \frac{\partial u}{\partial y} + F(x, y, t) u = H(x, y, t) \quad (2.1)$$

with appropriate initial and boundary conditions. If  $Q(x) \equiv 0$  and the functions  $A, B, C, D, E, F$ , and  $H$  all lack time dependence, then `coll2d.c` solves the PDE

$$\mathcal{L}[u](x, y) = A(x, y) \frac{\partial^2 u}{\partial x^2} + B(x, y) \frac{\partial^2 u}{\partial x \partial y} + C(x, y) \frac{\partial^2 u}{\partial y^2} + D(x, y) \frac{\partial u}{\partial x} + E(x, y) \frac{\partial u}{\partial y} + F(x, y) u = H(x, y) \quad (2.2)$$

with appropriate boundary conditions.

The collocation discretization of (2.2) (respectively, (2.1)) is completely analogous to that of (1.2) (respectively, (1.1)). Therefore, we will discuss the particulars of two-dimensional collocation and not repeat the details of the derivation that carry over from the one-dimensional problem in an obvious manner. To this end, we will focus on the PDE (2.2).

## 2.2 Collocation in two dimensions

### 2.2.1 Hermite interpolating polynomial

Here we let  $u(x, y)$  be a function defined on the rectangular domain  $\mathcal{D} = [a_x, b_x] \times [a_y, b_y]$ . If we define a set  $X = \{a_x = x_0 < x_1 < x_2 < \dots < x_{m_x} = b_x\}$  for the  $x$ -direction and a set  $Y = \{a_y = y_0 < y_1 < y_2 < \dots < y_{m_y} = b_y\}$  for the  $y$ -direction, then the set of points

$$\{(x_q, y_r) : x_q \in X \text{ and } y_r \in Y\}$$

defines the set of nodes for our two-dimensional problem. Clearly, this partitions the domain  $\mathcal{D}$  into  $m_x m_y$  rectangular finite elements, where  $m_x$  is the number of elements in the  $x$ -direction and  $m_y$  (assumed to be even throughout this work) is the number of elements in the  $y$ -direction. By analogy to (1.6), the piecewise bi-cubic polynomial interpolating  $u_{q,r} = u(x_q, y_r)$ ,  $u_{q,r}^x = \frac{du}{dx}(x_q, y_r)$ ,  $u_{q,r}^y = \frac{du}{dy}(x_q, y_r)$ , and  $u_{q,r}^{xy} = \frac{d^2u}{dx dy}(x_q, y_r)$  for  $q = 0, 1, 2, \dots, m_x$  and  $r = 0, 1, 2, \dots, m_y$  is

$$\hat{u}(x, y) = \sum_{q=0}^{m_x} \sum_{r=0}^{m_y} \left[ u_{q,r} f_q(x) f_r(y) + u_{q,r}^x g_q(x) f_r(y) + u_{q,r}^y f_q(x) g_r(y) + u_{q,r}^{xy} g_q(x) g_r(y) \right]. \quad (2.3)$$

A depiction of the four degrees of freedom  $u_{q,r}$ ,  $u_{q,r}^x$ ,  $u_{q,r}^y$ ,  $u_{q,r}^{xy}$  at each mesh point  $(x_q, y_r)$  for  $q = 0, 1, 2, \dots, m_x$  and  $r = 0, 1, 2, \dots, m_y$  on the domain  $\mathcal{D}$  is given in Figure 2.1 for the specific case  $m_x = 3$  and  $m_y = 4$ .

### 2.2.2 Collocation points

The collocation discretization proceeds by introducing (2.3) into (2.2), resulting in

$$\mathcal{L}[\hat{u}](x, y) - H(x, y) = E(x, y),$$

where  $E(x, y)$  is an error function. We then enforce

$$E(x, y) = 0 \quad (2.4)$$

at four distinct collocation points within each rectangular finite element. Since there are  $m_x m_y$  finite elements, we are defining  $4m_x m_y$  equations by enforcing (2.4) at the collocation points.

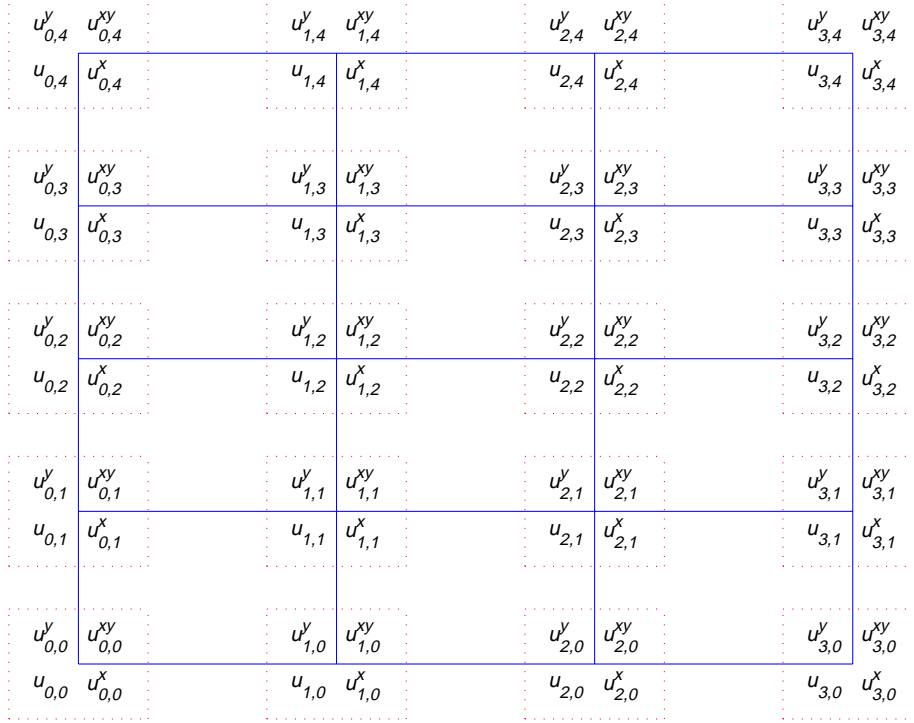


Figure 2.1: Location of degrees of freedom for two-dimensional Hermite interpolation

With reference to (1.4) and (1.5), we define the collocation point locations in terms of the parameter  $\eta$ . With reference to (2.3), it is clear that each collocation point requires a definition of  $\eta_x$  for the  $x$ -coordinate and a (possibly) independent definition of  $\eta_y$  for the  $y$ -coordinate. In the `coll2d.c` software, one chooses a parameter  $\xi \in (0, \frac{1}{2})$  for each finite element. Then the four collocation points within each finite element are defined as

$$(\eta_x, \eta_y) = \begin{cases} (-\xi, -\xi) \\ (-\xi, \xi) \\ (\xi, -\xi) \\ (\xi, \xi) \end{cases}$$

Given certain smoothness conditions, the value of  $\xi$  that provides maximum accuracy is  $\xi = \frac{1}{\sqrt{12}}$  [8]. The effect upon speed of convergence induced by varying  $\xi$  for problems in two spatial dimensions is discussed in detail in [1].

### 2.2.3 Boundary conditions

The proper definition of boundary conditions in two-dimensional collocation is much more complicated than in its one-dimensional counterpart. For the purposes of discussion, let us assume that Dirichlet boundary conditions are specified on the west boundary (i.e.,  $u(a_x, y)$  is given) and on the south boundary (i.e.,  $u(x, a_y)$  is given). Let us also assume that Neumann boundary conditions are specified on the east boundary (i.e.,  $\frac{\partial u}{\partial x}(b_x, y)$  is given) and on the north boundary (i.e.,  $\frac{\partial u}{\partial y}(x, b_y)$  is given).

Since the function  $u(a_x, y)$  is known, we may differentiate with respect to  $y$  to obtain  $\frac{\partial u}{\partial y}(a_x, y)$ . Similarly, we may differentiate the other known boundary condition functions to obtain  $\frac{\partial u}{\partial x}(x, a_y)$ ,  $\frac{\partial^2 u}{\partial x \partial y}(b_x, y)$ , and  $\frac{\partial^2 u}{\partial x \partial y}(x, b_y)$ . Thus at the west (W) boundary nodes we know  $u$  and  $\frac{\partial u}{\partial y}$ ; on the south (S) boundary nodes we know  $u$  and  $\frac{\partial u}{\partial x}$ ; on the east (E) boundary nodes we know  $\frac{\partial u}{\partial x}$  and  $\frac{\partial^2 u}{\partial x \partial y}$ ; and on the north (N) boundary nodes we know  $\frac{\partial u}{\partial y}$  and  $\frac{\partial^2 u}{\partial x \partial y}$ .

Now that all conditions on the *edges* of the boundaries are well-understood, we turn our attention to the four *corners*. It is clear that at the southwest (SW) boundary corner we know  $u$  and  $\frac{\partial u}{\partial x}$  from the S boundary condition and  $u$  and  $\frac{\partial u}{\partial y}$  from the W boundary condition. Thus the only degree of freedom that remains unknown at the SW corner node is  $\frac{\partial^2 u}{\partial x \partial y}$ . Similarly, we readily see that  $\frac{\partial u}{\partial y}$  is unknown at the southeast (SE) corner node, that  $u$  is unknown at the northeast (NE) corner node, and that  $\frac{\partial u}{\partial x}$  is unknown at the northwest (NW) corner node.

Since we know three degrees of freedom at each of the four corner nodes and two degrees of freedom at each boundary node not located at a corner of  $\mathcal{D}$ , a bit of arithmetic shows that there are a total of  $4m_x m_y$  unknown degrees of freedom. Since we also have  $4m_x m_y$  equations (generated by enforcing (2.4) at the collocation points), we are able to determine the values of the  $4m_x m_y$  unknowns.

## 2.3 Numbering of equations and unknowns

Collocation provides freedom in how one orders the equations and unknowns. We will first discuss the ordering apparently introduced in [5] and then re-discovered in [6]. We then derive the red-black ordering of [1].

$V_{42}=U_{0,4}^x$	$V_{43}=U_{1,4}$	$V_{44}=U_{1,4}^x$	$V_{45}=U_{2,4}$	$V_{46}=U_{2,4}^x$	$V_{47}=U_{3,4}$
42	43	44	45	46	47
31	33	35	37	39	41
$V_{31}=U_{0,3}^{xy}$	$V_{33}=U_{1,3}^y$	$V_{35}=U_{1,3}^{xy}$	$V_{37}=U_{2,3}^y$	$V_{39}=U_{2,3}^{xy}$	$V_{41}=U_{3,3}^y$
$V_{30}=U_{0,3}^x$	$V_{32}=U_{1,3}$	$V_{34}=U_{1,3}^x$	$V_{36}=U_{2,3}$	$V_{38}=U_{2,3}^x$	$V_{40}=U_{3,3}$
30	32	34	36	38	40
19	21	23	25	27	29
$V_{19}=U_{0,2}^{xy}$	$V_{21}=U_{1,2}^y$	$V_{23}=U_{1,2}^{xy}$	$V_{25}=U_{2,2}^y$	$V_{27}=U_{2,2}^{xy}$	$V_{29}=U_{3,2}^y$
$V_{18}=U_{0,2}^x$	$V_{20}=U_{1,2}$	$V_{22}=U_{1,2}^x$	$V_{24}=U_{2,2}$	$V_{26}=U_{2,2}^x$	$V_{28}=U_{3,2}$
18	20	22	24	26	28
7	9	11	13	15	17
$V_{7}=U_{0,1}^{xy}$	$V_{9}=U_{1,1}^y$	$V_{11}=U_{1,1}^{xy}$	$V_{13}=U_{2,1}^y$	$V_{15}=U_{2,1}^{xy}$	$V_{17}=U_{3,1}^y$
$V_{6}=U_{0,1}^x$	$V_{8}=U_{1,1}$	$V_{10}=U_{1,1}^x$	$V_{12}=U_{2,1}$	$V_{14}=U_{2,1}^x$	$V_{16}=U_{3,1}$
6	8	10	12	14	16
0	1	2	3	4	5
$V_{0}=U_{0,0}^{xy}$	$V_{1}=U_{1,0}^y$	$V_{2}=U_{1,0}^{xy}$	$V_{3}=U_{2,0}^y$	$V_{4}=U_{2,0}^{xy}$	$V_{5}=U_{3,0}^y$

Figure 2.2: Preliminary numbering of equations and unknowns for two-dimensional collocation

### 2.3.1 The preliminary ordering scheme

Consider the numbering of equations and unknowns given in Figure 2.2 for the example  $m_x = 3$ ,  $m_y = 4$ . The locations of the collocation points, each of which represents one equation, are given by the magenta numbers in the interior of the finite elements. The unknowns, depicted in green, are each associated with a particular node in the finite element mesh. The matrix equation one solves to determine the unknowns may be written

$$\mathbf{M}\mathbf{v} = \mathbf{b}, \quad (2.5)$$

where, for the example  $m_x = 3$ ,  $m_y = 4$ , the matrix  $\mathbf{M}$  has the structure given in Figure 2.3. With respect to Figure 2.3, we see that each equation is associated with sixteen degrees of freedom because each collocation point “hits” the four nodes that define the corners of the finite element that contains it, and each of the four nodes is associated with four degrees of freedom.

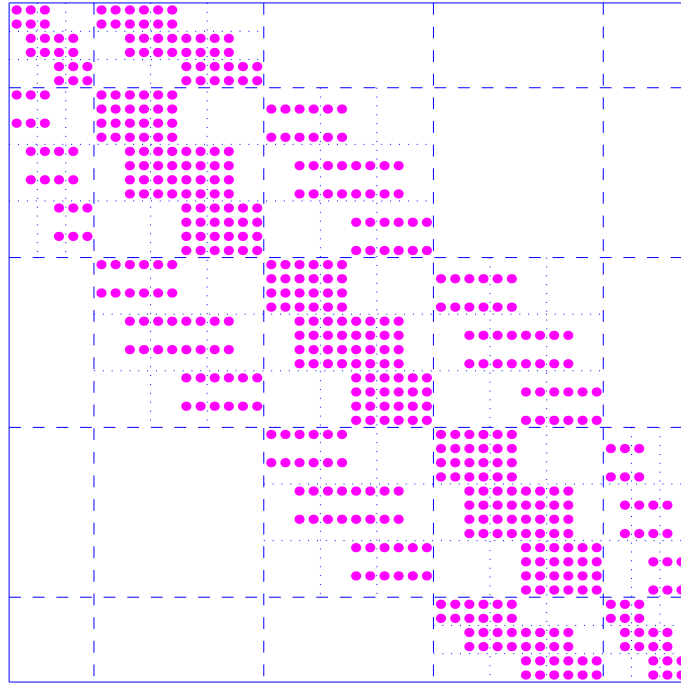


Figure 2.3: Structure of matrix  $\mathbf{M}$  arising from preliminary numbering scheme

The rows of matrix  $\mathbf{M}$  in Figure 2.3 are in the order given by the numbering of the collocation points in Figure 2.2 while the columns of matrix  $\mathbf{M}$  in Figure 2.3 are ordered by the subscripts  $j$  of  $v_j$  in Figure 2.2. Of course, many rows of matrix  $\mathbf{M}$  contain fewer than 16 unknowns: the “missing” degrees of freedom are found in the vector  $\mathbf{b}$  which contains this boundary condition information. The orderings of the entries of vectors  $\mathbf{v}$  and  $\mathbf{b}$  correspond to the orderings of the unknowns and collocation points, respectively.





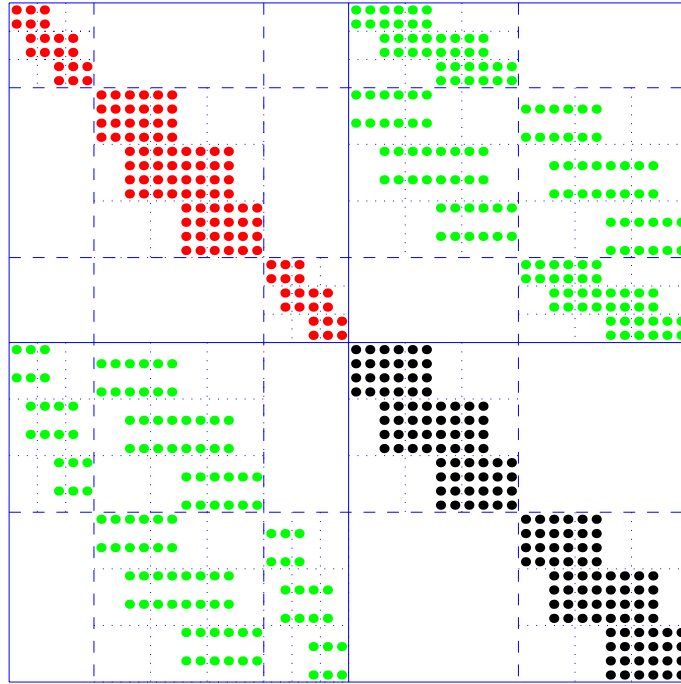


Figure 2.5: Matrix structure arising from Red-Black numbering of equations and unknowns

which we abbreviate

$$\left[ \begin{array}{c|c} \mathbf{R} & \mathbf{U} \\ \mathbf{L} & \mathbf{B} \end{array} \right] \begin{bmatrix} \mathbf{v}_R \\ \mathbf{v}_B \end{bmatrix} = \begin{bmatrix} \mathbf{b}_R \\ \mathbf{b}_B \end{bmatrix}. \quad (2.7)$$

## 2.4 Preconditioned conjugate gradient methods

We now discuss the solution of the linear system (2.7), which we elect to solve (2.7) using a preconditioned conjugate-gradient-type method. Since the matrix in (2.7) is neither symmetric nor, in general, diagonally dominant, we must choose a method that is particularly well suited to such matrices. The `col12d.c` software allows the user to select either the GMRES [11] method or the Bi-CGSTAB [12] method. Irrespective of which method is

chosen, the selection of a suitable preconditioning matrix is necessary for the rapid convergence we seek. In the `coll2d.c` software, the preconditioner is the matrix

$$\mathbf{P} = \left[ \begin{array}{c|c} \mathbf{R} & \\ \hline \mathbf{L} & \mathbf{B} \end{array} \right].$$

The preconditioned version of Bi-CGSTAB is described in [12]. For a description of preconditioned GMRES, see [2] or [10].

An important operation of both the GMRES and Bi-CGSTAB algorithms is the solution of linear systems of the form  $\mathbf{P}\mathbf{y} = \mathbf{c}$ , which we write

$$\left[ \begin{array}{c|c} \mathbf{R} & \\ \hline \mathbf{L} & \mathbf{B} \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_R \\ \mathbf{y}_B \end{array} \right] = \left[ \begin{array}{c} \mathbf{c}_R \\ \mathbf{c}_B \end{array} \right]. \quad (2.8)$$

To compute  $\mathbf{y} = \left[ \begin{array}{c} \mathbf{y}_R \\ \mathbf{y}_B \end{array} \right]$ , we first solve  $\mathbf{R}\mathbf{y}_R = \mathbf{c}_R$  for  $\mathbf{y}_R$ . That is, we solve

$$\left[ \begin{array}{cccccccc} \mathbf{A}_F & & & & & & & \\ & \mathbf{A}_1 & & & & & & \\ & & \mathbf{A}_3 & & & & & \\ & & & \ddots & & & & \\ & & & & \mathbf{A}_{m_y-3} & & & \\ & & & & & \mathbf{A}_L & & \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_F \\ \mathbf{y}_1 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_{m_y-3} \\ \mathbf{y}_L \end{array} \right] = \left[ \begin{array}{c} \mathbf{c}_F \\ \mathbf{c}_1 \\ \mathbf{c}_3 \\ \vdots \\ \mathbf{c}_{m_y-3} \\ \mathbf{c}_L \end{array} \right].$$

Because all the non-zero entries of  $\mathbf{R}$  are contained in its diagonal blocks, the solution of each subsystem  $\mathbf{A}_j\mathbf{y}_j = \mathbf{c}_j$  is an independent problem. Furthermore, when  $j = F$  or  $j = L$ , we see that  $\mathbf{A}_j$  is  $2 \times 2$  block tridiagonal and thus  $\mathbf{A}_j\mathbf{y}_j = \mathbf{c}_j$  is solved by precisely the method described in Section 1.4.3. When  $j = 0, 1, 2, \dots, m_y - 2$ , we see that  $\mathbf{A}_j$  is  $4 \times 4$  block tridiagonal and thus  $\mathbf{A}_j\mathbf{y}_j = \mathbf{c}_j$  ( $j = 1, 3, 5, \dots, m_y - 3$ ) is solved by the same method adapted for the  $4 \times 4$  case.

Once we have computed the values of  $\mathbf{y}_R = \left[ \mathbf{y}_F \ \mathbf{y}_1 \ \mathbf{y}_3 \ \cdots \ \mathbf{y}_{m_y-3} \ \mathbf{y}_L \right]^T$ , we compute  $\mathbf{y}_B$  by

$$\mathbf{B}\mathbf{y}_B = \mathbf{c}_B - \mathbf{L}\mathbf{y}_R = \mathbf{c}_B^*.$$

That is, we solve

$$\left[ \begin{array}{cccccccc} \mathbf{A}_0 & & & & & & & \\ & \mathbf{A}_2 & & & & & & \\ & & \ddots & & & & & \\ & & & \mathbf{A}_{m_y-4} & & & & \\ & & & & \mathbf{A}_{m_y-2} & & & \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_0 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{m_y-4} \\ \mathbf{y}_{m_y-2} \end{array} \right]$$



named block of matrix  $\mathbf{M}$  in (2.6) (see also Figure 2.3) has a block tridiagonal structure, where the blocks are either  $2 \times 2$  matrices,  $2 \times 4$  matrices, or  $4 \times 4$  matrices.

The numbering system of a `struct triblock22` is found in (1.26) for the case  $m_x = 3$ .

Here is the numbering system of a `struct triblock24`  $G_{24}$  for the case  $m_x = 3$ :

$$G_{24} = \left[ \begin{array}{cc|cc|cc} G_{24.a.0.0} & G_{24.a.0.1} & G_{24.b.0} & & & \\ \hline & G_{24.c.0} & G_{24.a.1.0} & G_{24.a.1.1} & G_{24.b.1} & \\ \hline & & & G_{24.c.1} & G_{24.a.2.0} & G_{24.a.2.1} \end{array} \right]$$

where each named block is a `struct m22` (call it  $M_{22}$ ) with numbering system

$$M_{22} = \left[ \begin{array}{cc} M_{22.00} & M_{22.01} \\ M_{22.10} & M_{22.11} \end{array} \right].$$

The numbering system of a `struct triblock44`  $G_{44}$  for the case  $m_x = 3$  is

$$G_{44} = \left[ \begin{array}{cc|cc|cc} G_{44.a.0.A} & G_{44.a.0.B} & G_{44.b.0.0} & & & \\ G_{44.a.0.C} & G_{44.a.0.D} & G_{44.b.0.1} & & & \\ \hline & G_{44.c.0.0} & G_{44.a.1.A} & G_{44.a.1.B} & G_{44.b.1.0} & \\ & G_{44.c.0.1} & G_{44.a.1.C} & G_{44.a.1.D} & G_{44.b.1.0} & \\ \hline & & & G_{44.c.1.0} & G_{44.a.2.A} & G_{44.a.2.B} \\ & & & G_{44.c.1.0} & G_{44.a.2.C} & G_{44.a.2.D} \end{array} \right]$$

where each named block is a `struct m22`.

- `struct collvec` — represents the storage required to hold the entries of a vector  $\mathbf{v}$  or  $\mathbf{b}$  in (2.5) or (2.6). A `struct collvec` consists of two `struct kvec2s` (see Section 1.4.1) (for storage of, for example,  $\mathbf{v}_F$  and  $\mathbf{v}_L$ ) and  $m_y - 1$  `struct kvec4s` (for storage of, for example,  $\mathbf{v}_j$  ( $j = 0, 1, \dots, m_y - 2$ )). A single `struct kvec4`  $k$  is numbered

$$k = \left[ \begin{array}{cccc} k.0 & k.1 & \cdots & k.m_x - 1 \end{array} \right]$$

where each named block is a `struct v4` (call it  $v$ ) with numbering system

$$v = \left[ \begin{array}{cccc} v.0.0 & v.0.1 & v.1.0 & v.1.1 \end{array} \right].$$

- `struct corner_element_bcs` — holds boundary values for a single corner element. It contains 32 entries: Each corner element has four equations, each of which contains boundary value information on two sides of  $\mathcal{D}$  (an E or W side and a N or S side). On each of these sides there are two nodes (on the boundary of the finite element) and each node contains two known degrees of freedom from boundary conditions. If we recall the discussion in Section 2.2.3 about boundary conditions at corners of  $\mathcal{D}$ , we see that a `struct corner_element_bcs` contains four pieces of redundant information, one for each equation in the corner finite element under consideration.

- `struct EW_element_bcs` — holds boundary values for a single element along the E or W boundary of  $\mathcal{D}$ . It contains  $16(m_y - 2)$  entries: There are  $m_y - 2$  elements along the E and W boundaries that are not corner elements. Each of them contains four equations, each of which contains boundary value information on one sides of  $\mathcal{D}$  (an E or W side). On the particular side there are two nodes (on the boundary of the finite element) and each node contains two known degrees of freedom from boundary conditions.

- `struct NS_element_bcs` — holds boundary values for a single element along the N or S boundary of  $\mathcal{D}$ . It contains  $16(m_x - 2)$  entries: There are  $m_x - 2$  elements along the N and S boundaries that are not corner elements. Each of them contains four equations, each of which contains boundary value information on one sides of  $\mathcal{D}$  (a N or S side). On the particular side there are two nodes (on the boundary of the finite element) and each node contains two known degrees of freedom from boundary conditions.

## 2.6.2 The file `coll2d.c`

This file contains the commands that solve the PDEs (2.1) or (2.2). It also calls subroutines contained in the files `fun2d_p1.h`, `fun2d_p2.h`, and `fun2d_p3.h`. We will proceed through the file `coll2d.c`, discussing salient features of the software.

### Parameters for problem definition

For those who want to use `coll2d.c` as a “black box” solver, we will first describe those parts of the code that define a specific problem. All are analogous to similar statements in the `coll1d.c` code.

- `#define xm, ym` (lines 6-7). These lines defines the number of finite elements  $m_x$  in the  $x$ -direction (`xm`) and the number of finite elements  $m_y$  in

the  $y$ -direction (`ym`).

- `#define sstol` (line 8). See Section 1.4.2.

- `#define A(x,y,t), B(x,y,t), C(x,y,t), D(x,y,t), E(x,y,t), F(x,y,t)` and `Q(x,y)` (lines 19-25). These functions are found in the PDE (2.1). If solving the PDE (2.2), we require  $Q(x) \equiv 0$  and the functions  $A$ ,  $B$ ,  $C$ ,  $D$ ,  $E$ , and  $F$  to be independent of  $t$ .

- `#define S(x,y,t), S_x(x,y,t), S_y(x,y,t), and S_xy(x,y,t)` (lines 30-33). The solution of the PDE (2.1) requires appropriate initial conditions for the functions  $u(x,y,t)$ ,  $\frac{\partial u}{\partial x}(x,y,t)$ ,  $\frac{\partial u}{\partial y}(x,y,t)$ , and  $\frac{\partial^2 u}{\partial x \partial y}(x,y,t)$ , respectively. Use a fixed initial value of  $t = \text{init\_t}$  (see Section 1.4.2).

- `#define H(x,y,t)` (line 41). See Section 1.4.2.

- `#define BCTypeS, BCTypeN, BCTypeW, BCTypeE` (lines 46-49). The code handles two types of boundary conditions. A Type 1 (or Dirichlet) boundary condition specifies  $u$  at along a side of the boundary while a Type 0 (or Neumann) boundary condition specifies  $\frac{\partial u}{\partial x}$  along the E and W sides or  $\frac{\partial u}{\partial y}$  along the N and S sides.

- `#define BCValu...` (lines 54-61). These functions specify the formulas for the boundary conditions. If solving the PDE (2.1), these functions may be defined to depend on  $t$ . If solving the PDE (2.2), these functions must be independent of  $t$ . Note the discussion in Section 2.2.3 about obtaining additional boundary information by differentiating the given boundary value functions. The `coll2d.c` code relies on the user to analytically perform the four required differentiations.

- `#define restol` (lines 125). Convergence criterion which halts Bi-CGSTAB or GMRES iterations. Convergence is defined by the norm of the residual vector being less than `restol`. The 2-norm is used for GMRES while the  $\infty$ -norm is used for Bi-CGSTAB. The 2-norm is the natural norm to use for GMRES as it requires no additional computation. To use the 2-norm for Bi-CGSTAB, simply replace line 1080 in `fun2d_p3.h` with

```
*nm = norm2_collvec(&r);
```

- `/* define nodes */` (lines 159-168). Here we define the nodes  $a_x = x_0 < x_1 < x_2 < \dots < x_{m_x} = b_x$  and  $a_y = y_0 < y_1 < y_2 < \dots < y_{m_y} = b_y$ . We require  $m_x + 1$  nodes in the  $x$ -direction and  $m_y + 1$  nodes in the  $y$ -direction

- `/* define xi values */` (lines 174-176). See Sections 1.4.2 and 2.2.2. The code allows  $\xi$  to assume a different value on each finite element. Presently, however,  $\xi$  is set to the Gaussian value of  $\frac{1}{\sqrt{12}}$  for all finite elements.

- `/* time control */` (lines 181-184). See Section 1.4.2.

## Description of the code

The `coll2d.c` code follows the same sequence of steps as does the `coll1d.c` code. As a result, we will comment only upon those parts of the `coll2d.c` code that warrant special attention. Otherwise, refer to the discussion in Section 1.4.2.

- `#define method` (line 10). Two different algorithms are available to solve the system of linear algebraic equations (2.7): Bi-CGSTAB (method 1) and GMRES (method 2).

- `#define restart` (line 11). The definition of a restart parameter is necessary for the efficient implementation of GMRES (see [11] for discussion). The value of this parameter is presently set to 20.

- `#define pr` (line 12). Both Bi-CGSTAB and GMRES are iterative methods. If you want to see the behavior of the residuals from iteration to iteration, set `pr` to 1; otherwise, set `pr` to 0.

- `#define L_...` (lines 78-96). In contrast to the one-dimensional case where there are only four basis functions per linear finite element (see Figure 1.1), we have, in the two-dimensional case, sixteen basis functions per rectangular finite element. Sixteen basis functions are required because each collocation equation is associated with sixteen degrees of freedom (four degrees of freedom at each of four corner nodes in a particular finite element). The definitions of `L_...` represent the differential operator  $\mathcal{L}$  (see (2.2)) applied to each of the sixteen basis functions. (Of course, the differential operator  $\mathcal{L}$  may contain time dependence if applied to the PDE (2.1).)

- `#define J_...` (lines 99-117). For the two dimensional problem, the differential operator  $\mathcal{J}$  is defined

$$\mathcal{J}[u](x, y, t) = Q(x, y) u.$$

As with the differential operator  $\mathcal{L}$ , sixteen versions of `J_...` (i.e., differential operator  $\mathcal{J}$ ) must be defined.

- `#define maxiter` (line 119). Execution of the program ceases if the solver chosen (Bi-CGSTAB or GMRES) fails to converge in at most `maxiter` iterations.

- `#define xc, yc` (lines 121-122). Number of collocation points in the  $x$ - and  $y$ -directions, respectively.

degree of freedom	code
$u$	0
$\frac{\partial u}{\partial x}$	1
$\frac{\partial u}{\partial y}$	2
$\frac{\partial^2 u}{\partial x \partial y}$	3

Table 2.1: Coding for corner node unknowns

- `BCind` (line 236). It is convenient to keep track of which of the four degrees of freedom at a corner node is the one that is unknown. Table 2.1 gives the coding for this. For example, given the example discussed in Section 2.2.3, we see that `unkSW` is set to 3, `unkSE` is set to 2, `unkNE` is set to 0, and `unkNW` is set to 1.

- `/* results to screen */` (lines 370-375). Prints to the screen the time step number  $n + 1$ , the value of  $t_{n+1}$ , and the value of  $\Delta t = t_{n+1} - t_n$ . It then prints a table, the columns of which are, for node  $(x_q, y_r)$ :  $x_q$ ,  $y_r$ ,  $u_{qr}$ ,  $u_{qr}^x$ ,  $u_{qr}^y$ , and  $u_{qr}^{xy}$  for  $q = 0, 1, \dots, m_x$ ,  $r = 0, 1, \dots, m_y$ .

### 2.6.3 The file `fun2d_p1.h`

We discuss here certain subroutines of interest. Most are self-explanatory. We will focus on those that are not.

- `seam` (lines 661-673). Let us revisit equations (2.5) and (2.6). However, let us consider these equations in the context of multiplying a known matrix  $\mathbf{M}$  by a known vector  $\mathbf{v}$  to obtain vector  $\mathbf{b}$ . A typical row in this matrix-vector multiplication is

$$\mathbf{C}_{i-1}\mathbf{v}_{i-1} + \mathbf{A}_i\mathbf{v}_i + \mathbf{B}_i\mathbf{v}_{i+1} = \mathbf{b}_i.$$

Assume the entries of the vectors  $\mathbf{v}_j$  are indexed  $j = 0, 1, \dots, 4m_x - 1$ . If we consider the structure of the matrices  $\mathbf{C}_{i-1}$  and  $\mathbf{B}_i$  in Figure 2.3, we see that vector  $\mathbf{C}_{i-1}\mathbf{v}_{i-1}$  has all its *odd* indexed entries equal to zero while the vector  $\mathbf{B}_i\mathbf{v}_{i+1}$  has all of its *even* indexed entries equal to zero. Thus  $\mathbf{C}_{i-1}\mathbf{v}_{i-1}$  and  $\mathbf{B}_i\mathbf{v}_{i+1}$  each require a `struct kvec2` for storage (as opposed to a `struct`

`kvec4` as required for  $\mathbf{v}_j$ ). Because of the odd-even offset of the indexing, to add  $\mathbf{C}_{i-1}\mathbf{v}_{i-1} + \mathbf{B}_i\mathbf{v}_{i+1}$  requires nothing more than a “merging” or “seaming” of the vectors  $\mathbf{C}_{i-1}\mathbf{v}_{i-1}$  and  $\mathbf{B}_i\mathbf{v}_{i+1}$ .

#### 2.6.4 The file `fun2d_p2.h`

- `mult_collvec_collmat_trick` (lines 861-897). During the iteration process in the Bi-CGSTAB algorithm, we note that every step of the form (2.8)

$$\mathbf{P}\mathbf{y} = \mathbf{c}$$

is immediately followed by a matrix-vector multiplication step of the form

$$\mathbf{v} = \mathbf{A}\mathbf{y}.$$

Because of the manner in which  $\mathbf{P}$  was selected, we see that we do not have to perform the complete matrix-vector multiplication  $\mathbf{v} = \mathbf{A}\mathbf{y}$ :

$$\begin{aligned} \mathbf{v} &= \mathbf{A}\mathbf{y} = \begin{bmatrix} \mathbf{R} & \mathbf{U} \\ \mathbf{L} & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_B \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R} \\ \mathbf{L} & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_B \end{bmatrix} + \begin{bmatrix} \mathbf{U} \\ \end{bmatrix} \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_B \end{bmatrix} \\ &= \mathbf{P}\mathbf{y} + \begin{bmatrix} \mathbf{U}\mathbf{y}_B \\ \end{bmatrix} \\ &= \mathbf{c} + \begin{bmatrix} \mathbf{U}\mathbf{y}_B \\ \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{c}_R + \mathbf{U}\mathbf{y}_B \\ \mathbf{c}_B \end{bmatrix} \end{aligned}$$

Thus, with our choice of  $\mathbf{P}$ , the matrix-vector multiplication steps in the Bi-CGSTAB algorithm are performed relatively inexpensively. To be precise, instead of performing the four multiplications  $\mathbf{R}\mathbf{y}_R$ ,  $\mathbf{U}\mathbf{y}_B$ ,  $\mathbf{L}\mathbf{y}_R$ , and  $\mathbf{B}\mathbf{y}_B$  and the two additions  $\mathbf{R}\mathbf{y}_R + \mathbf{U}\mathbf{y}_B$  and  $\mathbf{L}\mathbf{y}_R + \mathbf{B}\mathbf{y}_B$ , we need perform only one multiplication ( $\mathbf{U}\mathbf{y}_B$ ) and one addition ( $\mathbf{c}_R + \mathbf{U}\mathbf{y}_B$ ).

- `psolve` (lines 987-1037). During the iteration process of GMRES and Bi-CGSTAB, we have to solve equations of the form (2.8). This routine follows the steps discussed in Section 2.4.

### 2.6.5 The file `fun2d_p3.h`

- `sub_BC` (lines 828-1001). Consider the analogue of (1.22) and (1.24) for the two-dimensional problem. The parameter `p` in `sub_BC` represents  $\tau$  in the analogue of (1.22) and  $\bar{\tau}$  in the analogue of (1.24). The parameter `fact` in `sub_BC` represents  $-1$  in the analogue of (1.22) and  $1$  in the analogue of (1.24).

The only truly cryptic lines of code in the entire software occur in this subroutine. For example, consider line 849 (and its analogues on lines 870, 891, and 913):

```
if (unkSW - BCTypeW == 2)
```

Recall the discussion concerning redundant boundary condition information being stored (see the discussion re: `struct corner_element_bcs` in Section 2.6.1). This cryptic line of code ensures that the final piece of boundary condition information to be subtracted from the given vector is the proper one, avoiding the improper choice of using the redundant information twice.

# Bibliography

- [1] S. H. Brill, *The Solution of Two-Dimensional Partial Differential Equations via Hermite Collocation with Block Red-Black Gauss-Seidel Preconditioner*, Ph.D. Thesis, University of Vermont, 1998.
- [2] P. Chin and P. A. Forsyth, "A Comparison of GMRES and CGSTAB Accelerations for Incompressible Navier-Stokes Problems," *Journal of Computational and Applied Mathematics*, Vol. 46, pp. 415-426, 1993.
- [3] C. deBoor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.
- [4] C. deBoor and B. Swartz, "Collocation at the Gaussian Points," *SIAM J. Numer. Anal.*, Vol. 10, No. 4, pp. 582-606, 1973.
- [5] E. O. Frind and G. F. Pinder, "A Collocation Finite Element Method for Potential Problems in Irregular Domains," *International Journal for Numerical Methods in Engineering*, Vol. 14, pp. 681-701, 1979.
- [6] Y.-L. Lai, A. Hadjidimos, E. N. Houstis, and J. R. Rice, "On the Iterative Solution of Hermite Collocation Equations," *SIAM J. Matrix Anal. Appl.*, Vol. 16, No. 1, pp. 254-277, 1995. (also Technical Report CSD-TR-92-094, Purdue Univ., 1992)
- [7] P. M. Prenter, *Splines and Variational Methods*, John Wiley & Sons, Inc., New York, 1975.
- [8] P. M. Prenter and R. D. Russell, "Orthogonal Collocation for Elliptic Partial Differential Equations," *SIAM J. Numer. Anal.*, Vol. 13, No. 6, pp. 923-939, 1976.
- [9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.

- [10] Y. Saad and M. H. Schultz, *Parallel Implementation of Preconditioned Conjugate Gradient Methods*, Research Report YALEU/DCS/RR-425, Yale University, 1985.
- [11] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, Vol. 7, No. 3, pp. 856-869, 1986.
- [12] H. A. van der Vorst, "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, Vol. 13, No. 2, pp. 631-644, 1992.

# Appendix A

## Source Code for Collocation Software

The pages that follow contain the text of the source code for `coll1d.c`, `coll2d.c`, and their auxiliary files. A list of files and their respective number of pages is given in Table A.1.

file	number of pages
<code>coll1d.c</code>	5
<code>struct1d.h</code>	1
<code>fun1d.h</code>	7
<code>coll2d.c</code>	7
<code>struct2d.h</code>	2
<code>fun2d_p1.h</code>	13
<code>fun2d_p2.h</code>	20
<code>fun2d_p3.h</code>	27

Table A.1: Files for collocation software