A PARALLEL COLLOCATION METHOD FOR TWO DIMENSIONAL LINEAR PARABOLIC SEPARABLE PARTIAL DIFFERENTIAL EQUATIONS

By

Yong Wang

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
AUGUST, 1994

ISBN 0-315-98953-X

Canada

Name WANG YONG

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Mathematics
SUBJECT TERM

0405
SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

DALHOUSIE UNIVERSITY

FACULTY OF GRADUATE STUDIES

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "A Parallel Collocation Method for Two Dimensional Linear Parabolic Partial Differential Equations"

by Yong Wang

in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dated August 22, 1994

External Examiner

Research Supervisor

Examining Committee

DALHOUSIE UNIVERSITY

Date: **August, 1994**

Author: **Yong Wang**

Title: **A Parallel Collocation Method for Two Dimensional Linear Parabolic Separable Partial Differential Equations**

Department: **Mathematics, Statistics and Computing Science**

Degree: **Ph.D.** Convocation: **October** Year: **1994**

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

献给我的艾妻

To my wife

Contents

List of Tables

List of Figures

Abstract

In this thesis, an orthogonal spline collocation algorithm is formulated and implemented within a method-of-lines approach for the numerical solution of a class of linear parabolic partial differential equations. A detailed implementation of the orthogonal spline collocation algorithm for solving elliptic partial differential equations (PDEs) is given and the efficiency of the implementation is discussed. By collocating the elliptic PDEs at Gaussian points, a linear system in tensor product form is obtained. A matrix decomposition approach is used to reduce the linear system to a collection of independent almost block diagonal linear systems, each of which is solved by the almost block diagonal linear system solver ARCECO. The same approach is then applied within a method-of-lines context to solve partial differential equations of parabolic type. By collocating the parabolic PDE at Gaussian points, systems of ordinary differential equations (ODES) are generated. The Jacobian matrix of the ODE system has an almost block diagonal structure. This ODE system is solved using the differential/algebraic solver DASSL, which is modified to take advantage of the special structure of the Jacobian matrix. These algorithms are efficiently implemented in a parallel architecture, using an eight processor Alliant/FX2800. Numerical experiments are presented to demonstrate the parallel performance of these algorithms.

Acknowledgements

This thesis was written under the supervision of Professor Patrick Keast in Dalhousie University. The author would like to thank Professor Patrick Keast for his guidance. Without his help, the completion of this thesis would never have been possible.

The author would like to thank Professors Bernard Bialecki, Paul Muir and John Clements for their careful reading and valuable suggestions and comments. The author would like to thank the Faculty of Graduate Studies for the financial support.

The author is especially grateful to his wife, Eileen Lee for her understanding and constant encouragement.

Finally, the author would like to express his appreciation to his parents for their support.

Chapter 1

Introduction and Background Material

1.1 Introduction

Recent rapid developments in parallel computer architectures are having dramatic effects on the scale of scientific computations which are now possible. The utilization of special parallel algorithms to take advantage of such architectures is clearly important. In this thesis, we discuss the development of a method-of-lines, parallel, collocation algorithm for solving time dependent linear separable parabolic partial differential equations (PDEs) in two space dimensions.

There are three principal components in the method-of-lines approach for solving two dimension linear separable parabolic PDEs: (i) a scheme for the discretization of the spatial variables; (ii) an ordinary differential equation (ODE) solver used for the system of ODEs in time which results from the discretization; (iii) the linear algebra software which is chosen to take advantage of any special structure which is present in the linear systems which arise during the numerical solution of the ODE system.

For (i) we use orthogonal spline collocation at Gaussian points. The spatial dependence of the numerical solution is represented as a linear combination of basis functions. The discretization of the spatial dimensions is obtained by requiring the

numerical solution to satisfy the PDE at collocation points. These points are chosen to be the images of the Gauss points [12] on each subinterval into which the problem interval is partitioned by a given mesh. This still leaves open the choices of basis functions. For reasons of efficiency and flexibility in order and continuity we choose B-splines [14] as the basis functions.

For (ii) and (iii) the differential/algebraic solver DASSL [15], [59], is modified to solve the ODEs introduced by collocation. The Jacobian matrix of the resulting ODEs has an almost block diagonal (ABD) structure [19]. This leads us to modify DASSL to take advantage of this structure by using the almost block diagonal linear system solver ARCECO [20].

There have been many papers in the area of orthogonal spline collocation for solving differential equations, (see [25] for a comprehensive survey). In [12] de Boor and Swartz analyzed collocation methods for solving boundary value ODE problems. They showed that the collocation approximation can achieve a high order of convergence if the approximate solution is required to satisfy the differential equation at Gaussian points. In [42] Keast, Fairweather and Diaz made a computational study of finite element methods for second order linear boundary value ODE problems. They verified the orders of convergence of the collocation method by numerical experiments, and concluded that collocation is the cheapest method to solve a certain class of boundary value problems and the easiest to implement. In [4] COLSYS, a very successful implementation of collocation for solving boundary value ODEs, is described. This package implements orthogonal spline collocation with B-spline bases, and has proved to be very efficient for solving a variety of boundary value problems. Some research has been done on orthogonal collocation for solving elliptic PDEs, for example [37], but most of these authors have not considered parallel features in their implementation. Only recently Bialecki and Fairweather [11] introduced a matrix decomposition algorithm for solving separable partial differential equations of elliptic type which can be readily adapted for parallel computation. There exist efficient implementations of orthogonal spline collocation for solving one dimensional parabolic

PDEs, for example, PDECOL [51]. The package uses finite element collocation methods based on B-splines for the spatial discretization and a modification of Gear's method [31] for solving the ODEs. The systems of linear equations are solved using a band solver. In [43] a modified form of PDECOL, called EPDCOL, is described, which uses the more efficient linear equation solver COLROW [20], to take advantage of the block structure of the linear systems which arise.

A brief outline of the thesis is as follows. In Chapter 2, a description of ELLDCM, our implementation of Bialecki and Fairweather's matrix decomposition algorithm [11] for the solution of elliptic PDEs with simple Dirichlet boundary conditions is given, and its efficiency is discussed. As we will show later, much of the work done for the elliptic case will be incorporated into our later implementation for the parabolic case. Also in Chapter 2, a comparison of ELLDCM with Kaufman and Warner's code [41] is given. Kaufman and Warner's code solves two dimensional elliptic partial differential equations using the Rayleigh-Ritz-Galerkin method. In [11] Bialecki and Fairweather also proposed an algorithm for solving elliptic PDEs with general linear boundary conditions; we extend their approach in Chapter 2. In Chapter 3, we concentrate on the solution of PDEs of parabolic type. We develop and implement an orthogonal spline collocation algorithm, PARCOL, for solving two dimensional parabolic PDEs. By collocating the elliptic operator at Gaussian points, we obtain a system of ODEs. The differential/algebraic solver DASSL is employed to solve the ODE system. An almost block diagonal linear system solver ARCECO [20] is added to DASSL in order to efficiently handle the special structure of the Jacobian matrix which arises. The parallel performance of the algorithm is demonstrated through numerical tests on a parallel computer system, the Alliant/FX2800. In Chapter 4, we give the calling diagram of our programs, ELLDCM and PARCOL. For each of the codes, we present the initial comments in which we describe briefly the purpose and usage of each routine. We present our conclusions and future research directions in Chapter 5. In Appendix, we give sample driving programs for solving certain test problems using ELLDCM and PARCOL.

1.2 Background Material

In this section, we first examine the structure of the systems of linear equations which result from collocation with B-spline basis functions. The linear system which we will solve in Chapters 2 and 3 can be expressed in terms of a matrix tensor product; here we give the definition and some properties of this product. Also, we give the definition of the matrix Hadamard product which is used to simplify the notation in Chapter 3. Finally, we discuss some background material concerning parallel processing.

1.2.1 The Structure of the Collocation Matrix with B-spline bases

In order to illustrate the structure of the collocation matrix with B-spline basis functions, we give the collocation procedure for solving the following simple ODE problem:

$$\frac{d^2u}{dx^2} = f(x), \quad x \in [0, 1], \tag{1.1}$$

$$u(0) = u(1) = 0. \tag{1.2}$$

Let N be be a positive integer and let $\pi = \{x^{(j)}\}_{j=0}^{N}$ denote a partition of $[0, 1]$ such that

$$0 = x^{(0)} < x^{(1)} < \cdots < x^{(N)} = 1\,, \tag{1.3}$$

and $I^{(j)}$ denote the interval $[x^{(j-1)}, x^{(j)}]$, $j = 1, 2, \cdots, N$.

Let $r \geq 3$ and let $\mathcal{M}(r, \pi)$ be the space of piecewise polynomial functions defined by

$$\mathcal{M}(r, \pi) = \{v \in C^1[0, 1] : v|_{[x^{(j-1)}, x^{(j)}]} \in P_r\,, \quad j = 1, \ldots N, \quad v(0) = v(1) = 0\}, \tag{1.4}$$

where P_r denotes the set of all polynomials of degree $\leq r$. Note that we have chosen $\mathcal{M}(r, \pi)$ so that the boundary conditions (1.2) are satisfied by the whole space. The dimension of $\mathcal{M}(r, \pi)$, denoted by M, is $N(r-1)$. Let $\{\sigma^{(k)}\}_{k=1}^{r-1}$ and $\{w^{(k)}\}_{k=1}^{r-1}$ be

the nodes and weights of the $(r-1)$ point Gauss-Legendre quadrature rule on [0, 1]. The Gaussian points on $I^{(j)}$ are defined by

$$\xi^{((j-1)(r-1)+k)} = x^{(j-1)} + h^{(j)}\sigma^{(k)}, \quad j = 1, \ldots, N, \quad k = 1, \ldots, r-1, \tag{1.5}$$

where $h^{(j)} = x^{(j)} - x^{(j-1)}$.

The B-spline functions have been widely used for basis functions in collocation and finite element Galerkin methods for the numerical solution of differential equations. The definition and properties of B-splines can be found in [14]. B-splines are efficient representations of piecewise polynomial functions satisfying certain continuity conditions at a given set of mesh points. To describe the support properties of the B-splines, we must specify a set of knots $\{t_i\}$. These knot points are determined by the mesh selection, the continuity conditions and the order of the B-splines. For B-splines of order K (i.e. degree $\leq K-1$) with C^1 continuity on the mesh defined by (1.3), the knots are defined as follows:

$$\begin{aligned}
t_1 &= t_2 = \cdots = t_K = x^{(0)} \\
&< t_{K+1} = \cdots = t_{2K-2} = x^{(1)} \\
&< t_{2K-1} = \cdots = t_{3K-4} = x^{(2)} \\
&\cdot \\
&< t_{jK-2j+3} = \cdots = t_{(j+1)K-2j} = x^{(j)} \\
&\cdot \\
&< t_{(N-1)K-2(N-1)+3} = \cdots = t_{NK-2(N-1)} = x^{(N-1)} \\
&< t_{NK-2N+3} = t_{NK-2N+4} = \cdots = t_{(N+1)K-2N+2} = x^{(N)}.
\end{aligned}$$

The space of B-splines of order K with C^1 continuity conditions is denoted by S_2^K. The dimension of S_2^K is $N(K-2)+2$. The following lemmas from [14] are useful.

Lemma 1.1 *Let $\{\eta_j\}_{j=1}^K$ and $\{\psi_j\}_{j=1}^K$ denote the basis functions spanning the B-splines of order K, on the subintervals $I^{(1)}$ and $I^{(N)}$, respectively. Let $g^{(i)}(x)$ denote the ith derivative of the function $g(x)$. Then*

$$\eta_{j+1}^{(i)}(x)_{x=x_0^+} = l_{ij},$$

$$\psi_{j+1}^{(i)}(x)_{x=x_N^-} = u_{ij}, \quad i,j = 0,1,\cdots,K-1,$$

where

$$\begin{aligned} l_{ij} &= 0 \text{ if } i < j, \\ u_{ij} &= 0 \text{ if } K-i > j+1. \end{aligned}$$

Lemma 1.2 *Let $B_i(x)$, $i = 1,2,\cdots,N(K-2)+2$, denote the B-spline basis functions spanning S_2^K. Then the support of $B_i(x)$ is (t_i, t_{i+K}).*

From Lemma 1.1 we observe that to get a set of basis functions for the space $\mathcal{M}(r,\pi)$, we can just use B-spline basis functions for $\hat{\mathcal{M}}(r,\pi)$, where $\hat{\mathcal{M}}(r,\pi)$ is defined as

$$\hat{\mathcal{M}}(r,\pi) = \{v \in C^1[0,1] : v|_{[x_{j-1},x_j]} \in P_r, \ j = 1,\ldots N\}, \tag{1.6}$$

except that we "discard" the first basis function spanning the B-splines on the subinterval $I^{(1)}$ and the last basis function spanning the B-splines on the subinterval $I^{(N)}$. The remaining B-spline functions form a basis for $\mathcal{M}(r,\pi)$.

Let $\{\phi_n\}_{n=1}^M$ be a B-spline basis for $\mathcal{M}(r,\pi)$, where $M = N(K-2)$. Suppose that the approximate solution, $U \in \mathcal{M}(r,\pi)$, is given by

$$U(x) = \sum_{n=1}^{M} u_n \phi_n(x),$$

where u_n, $n = 1,\cdots,M$, are unknown coefficients to be determined. The orthogonal spline collocation solution for (1.1), (1.2) (that is collocation at the Gaussian points (1.5)) is obtained by requiring U to satisfy

$$\frac{d^2U}{dx^2}(\xi^{(m)}) = f(\xi^{(m)}), \quad m = 1,\ldots,M. \tag{1.7}$$

Note that U already satisfies the boundary conditions, which are imposed on $\mathcal{M}(r,\pi)$.

Let

$$\vec{u} = [u_1, u_2, \ldots, u_M]^T \text{ and } \vec{f} = [f_1, f_2, \ldots, f_M]^T,$$

where $f_m = f(\xi^{(m)})$. Then (1.7) can be written as

$$A\vec{u} = \vec{f}, \tag{1.8}$$

where

$$A = (a_{m\,n})_{m,n=1}^{M}, \text{ and } a_{m\,n} = \phi_n''(\xi^{(m)}).$$

From Lemma 1.2, the collocation matrix A is an almost block diagonal matrix (ABD) [19], with the structure shown in Fig 1.1. The linear system (1.8) can be solved efficiently by using the package ARCECO [20].

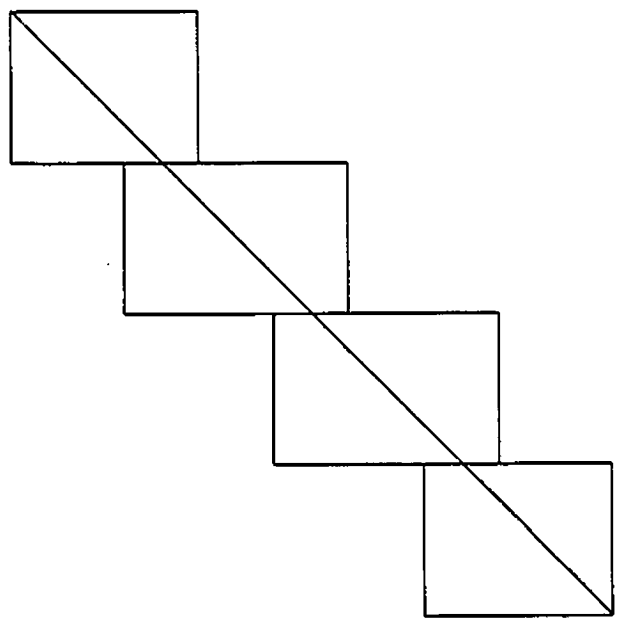

Figure 1.1: The structure of the collocation matrix.

In the above matrix, each block represents the collocation equations in one interval. The column overlap between blocks is determined by the continuity requirements. No three blocks overlap. Note that usually, for one dimensional boundary value problems (BVPs), $U(x) = \sum_{n=0}^{M+1} u_n B_n(x)$, where $\{B_n\}_{n=0}^{M+1}$ is the B-spline basis for

$\hat{\mathcal{M}}(r,\pi)$ of (1.6), boundary conditions such as $u(0) = u(1) = 0$ are collocated, i.e., $U(0) = U(1) = 0$, and these equations are included in the linear system. We have taken a slightly different approach here by making the basis functions satisfy the boundary conditions. This is required for the work in Chapter 2.

1.2.2 Order of Convergence

In [4], a method for spline collocation at Gaussian points is implemented for two point boundary value problems using B-splines. The package can solve a mixed order system of ODEs of the form:

$$\begin{aligned} u_i^{(m_i)} &= f_i(x, u_1, \cdots, u_1^{(m_1-1)}, u_2, \cdots, u_d^{(m_d-1)}) \\ &= f_i(x, \vec{z}(\vec{u})), \quad 1 \le i \le d, \quad a \le x \le b, \end{aligned} \tag{1.9}$$

with multi-point conditions

$$g_j(\zeta_j, \vec{z}(\vec{u})) = 0, \quad 1 \le j \le m^*, \tag{1.10}$$

where

$$\vec{u}(x) = (u_1(x), \cdots, u_d(x))^T, \quad m^* = \sum_{i=1}^{d} m_i, \quad a \le \zeta_1 \le \zeta_2 \cdots \le \zeta_{m^*} \le b,$$
$$\text{and} \quad \vec{z}(\vec{u}(x)) = (u_1(x), \cdots, u_1^{(m_1-1)}(x), u_2(x), \cdots, u_d^{(m_d-1)}(x))^T.$$

Given a partition of $[a,b]$, denoted by π, the approximate solution

$$\vec{v} = (v_1, v_2, \cdots, v_d),$$

is computed by requiring that $v_i \in P_{r_i,\pi} \cap C^{(m_i-1)}[a,b]$ $(r_i = k + m_i)$ and that v_i satisfy the differential equation (1.9) at k Gaussian points on each subinterval of π. As shown in [3] and [12], the error bounds of such approximations are, in general,

$$\| u_i^{(l)} - v_i^{(l)} \|_\infty = O(h^{k+m_i-l}), \quad l = 0, \cdots, m_i, \quad i = 1, \cdots, d, \tag{1.11}$$

and at mesh points,

$$|(u_i^{(l)} - v_i^{(l)})(x_j)| = O(h^{2k}), \quad i = 1, \cdots, d, \quad j = 0, \cdots, N, \quad l = 0, \cdots, m_i - 1. \tag{1.12}$$

Here h is the maximum subinterval length. By (1.11) the order of convergence for the L_∞ error for $v_i^{(l)}$ is $k + m_i - l$, for $i = 1, \cdots, d$ and $0 \leq l \leq m_i$. By (1.12) the order of convergence for the maximum error at mesh points for $v_i^{(l)}$ is $2k$, for $i = 1, \cdots, d$ and $0 \leq l \leq m_i$. The higher order of convergence at mesh points is called superconvergence.

An approximation to the order of convergence may be computed in the following way: If the error is $O(h^R)$, then the error $\approx c \cdot h^R$, where c is a constant independent of h. Let $ERROR_{h_1}$ and $ERROR_{h_2}$ denote the errors of two approximations with two different uniform meshes having subinterval sizes h_1 and h_2 and $h_1 > h_2$. Then $ERROR_{h_1} \approx c \cdot h_1^R$ and $ERROR_{h_2} \approx c \cdot h_2^R$, therefore

$$\frac{ERROR_{h_1}}{ERROR_{h_2}} \approx \left(\frac{h_1}{h_2}\right)^R,$$

hence

$$\log\left(\frac{ERROR_{h_1}}{ERROR_{h_2}}\right) \approx R \log(h_1/h_2).$$

Therefore the approximate order of convergence is given by

$$R \approx \frac{\log\left(ERROR_{h_1}/ERROR_{h_2}\right)}{\log\left(h_1/h_2\right)}. \tag{1.13}$$

Next we discuss the order of convergence of the interpolant which is obtained by interpolation at the Gaussian points to a given function in the presence of mixed linear boundary conditions. The result presented here will be used in the later discussion about the order of convergence of the orthogonal spline collocation algorithm applied to two dimensional elliptic PDEs with mixed linear boundary conditions.

Let the function u, defined on $[0, 1]$, be such that

$$\alpha_0 u(0) - \beta_0 u'(0) = c_0, \quad \alpha_1 u(1) - \beta_1 u'(1) = c_1. \tag{1.14}$$

Let $r \geq 3$ and $\hat{\mathcal{M}}(r, \pi)$ be the space of piecewise polynomial functions defined by (1.6). The dimension of $\hat{\mathcal{M}}(r, \pi)$ is $M + 2$, where $M = N(r - 1)$. Let U be an approximation to u such that $U \in \hat{\mathcal{M}}(r, \pi)$, U interpolates u at Gaussian points

defined as in (1.5), and also satisfies (1.14). Let $\{\phi_n(x)\}_{n=0}^{M+1}$ be a modified B-spline basis for $\hat{\mathcal{M}}(r,\pi)$ with ϕ_0, ϕ_1, ϕ_M and ϕ_{M+1} satisfying the conditions,

$$\alpha_0\phi_0(0) - \beta_0\phi_0'(0) = 1, \quad \alpha_0\phi_1(0) - \beta_0\phi_1'(0) = 0, \tag{1.15}$$

$$\alpha_1\phi_M(1) - \beta_1\phi_M'(1) = 0, \quad \alpha_1\phi_{M+1}(1) - \beta_1\phi_{M+1}'(1) = 1. \tag{1.16}$$

The remaining ϕ_i's satisfy (1.14) with zero right hand sides. Using Lemma 1.1, Bialecki and Fairweather [11] showed that bases satisfying equations (1.15) and (1.16) can be chosen. The details are given in Chapter 2.

Suppose that $U = \sum_{i=0}^{M+1} u_i\phi_i$. Then we have $\alpha_0U(0) - \beta_0U'(0) = c_0$ from (1.14). In addition

$$\begin{aligned}
&\alpha_0U(0) - \beta_0U'(0) = \alpha_0 \sum_{i=0}^{M+1} u_i\phi_i(0) - \beta_0 \sum_{i=0}^{M+1} u_i\phi_i'(0) \\
&= \alpha_0u_0\phi_0(0) - \beta_0u_0\phi_0'(0) + \alpha_0 \sum_{i=1}^{M+1} u_i\phi_i(0) - \beta_0 \sum_{i=1}^{M+1} u_i\phi_i'(0) \\
&= u_0, \text{ by Lemma 1.1 and equations (1.15).}
\end{aligned}$$

Therefore $u_0 = c_0$. Similarly we can show that $u_{M+1} = c_1$. The unknowns u_i, for $i = 1, \cdots, M$ can be determined from the linear system $A\vec{u} = \vec{f}$, where

$$\vec{u} = [u_1, \cdots, u_M]^T, \quad \vec{f} = [f_1, \cdots, f_M]^T, \; m = 1, \cdots, M,$$

where

$$f_m = u(\xi^{(m)}) - u_0\phi_0(\xi^{(m)}) - u_{M+1}\phi_{M+1}(\xi^{(m)}),$$

and $a_{mn} = \phi_n(\xi^{(m)})$, $m, n = 1, \cdots, M$, and $\xi^{(m)}$, $m = 1, \cdots, M$, is defined by (1.5).

As before for (1.1) and (1.2), the interpolant $U(x) = \sum_{n=0}^{M+1} u_nB_n(x)$ could be computed by collocating at the Gaussian points and at the end points, that is the two equations $\alpha_0U(0) - \beta_0U'(0) = c_0$, $\alpha_1U(1) - \beta_1U'(1) = c_1$ are included in the system. We have followed a different procedure by imposing certain conditions on the basis functions in order to prepare for the work in Chapter 2. In the following we show experimentally the order of convergence of this interpolation process. The results here are heuristic since they are only based on the observation on the output

of a numerical experiment. See Section 5.1 in [10] for theoretical results in case of Dirichlet boundary conditions with piecewise cubics. We choose u to be $x^6 - x^5 + 1$, and note that u satisfies

$$u(0) - u'(0) = 1, \quad u(1) - u'(1) = 0. \tag{1.17}$$

In Table 1.1, E_u denotes the L_∞ error on $[0,1]$, E_m denotes the maximum error at the mesh points, N is the number of subintervals, and k is the number of collocation points in each subinterval. The values R_u, R_m are computed according to (1.13) for consecutive mesh sizes. The L_∞ errors are computed by sampling the error at 100 uniformly distributed points. In Figure 1.2, we plot on a log-log scale the L_∞ errors and maximum errors at mesh points. Table 1.1 and Figure 1.2 show that the orders of convergence for both the L_∞ error and the maximum error at mesh points are approximately $k + 2$.

	k=2				k=3			
N	E_u	R_u	E_m	R_m	E_u	R_u	E_m	R_m
4	9.14-04		8.17-04		2.47-05		1.69-05	
8	6.97-05	3.71	5.90-05	3.79	8.22-07	4.91	5.59-07	4.92
16	4.80-06	3.86	3.95-06	3.90	2.65-08	4.95	1.79-08	4.97
32	2.55-07	4.24	2.56-07	3.95	8.41-10	4.98	5.67-10	4.98

Table 1.1: The errors of interpolation at Gaussian points.

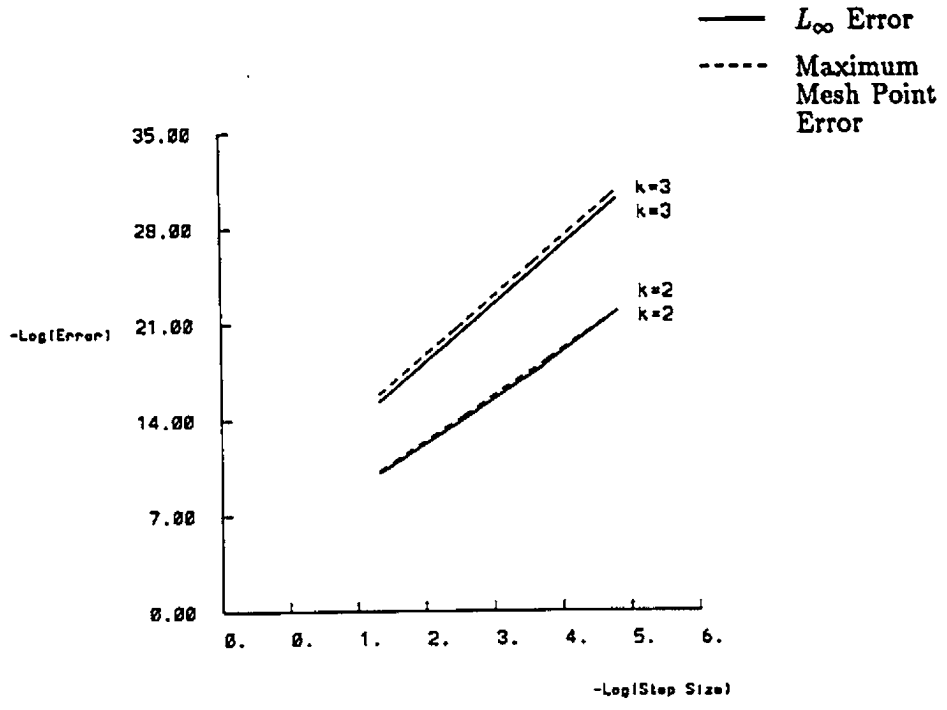

Figure 1.2: The order of convergence of function interpolation at Gaussian points.

The result shown here will be used for the discussion about the order of convergence of the algorithms given in Chapter 2 and 3. One should note that the order of convergence of the maximum error at mesh points is no higher than the order of convergence of the L_∞ error for the interpolation process discussed here. Although superconvergence at the mesh points is observed for collocation at Gaussian points, it is not observed for interpolation at Gaussian points.

1.2.3 Matrix Tensor Product and Hadamard Product

Let A and B be two matrices of size $m \times n$ and $p \times q$ respectively. The tensor product of A and B, denoted by $A \otimes B$, is defined by:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}.$$

Clearly $A \otimes B$ is a matrix of order $mp \times nq$. The following properties of the tensor product of the matrices will be frequently used ([53]).

(1) $(A \otimes B)(C \otimes D) = (AC \otimes BD)$.

(2) $(A + C) \otimes B = A \otimes B + C \otimes B$.

(3) $(A \otimes B)^T = A^T \otimes B^T$.

(4) If A and B are nonsingular, then $A \otimes B$ is nonsingular and

$$(A \otimes B)^{-1} = (A^{-1} \otimes B^{-1}).$$

Next we give the definition of the matrix Hadamard product. The Hadamard product of square matrices A and B of the same size $m \times m$, denoted by $A * B$, is defined by:

$$A * B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1m}b_{1m} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2m}b_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mm}b_{mm} \end{bmatrix}.$$

The definition and properties of matrix Hadamard product can be found in [53].

1.2.4 Parallel Processing

Parallel architectures have lead to the development of parallel algorithms which have challenged existing sequential algorithms for the numerical approximation of differential equations, and extensive work has been done on developing such algorithms, [56], [66], etc. The *speedup* of a parallel algorithm is defined as the ratio of execution time on a single processor to the execution time on a multiprocessor (see, eg; [46]). If we have an n processor system, then the *speedup* is defined as

$$speedup = \frac{t_1}{t_n}, \tag{1.18}$$

where t_1 is the execution time on a single processor and t_n is the execution time on n processors. Amdahl's law [46] states that

$$speedup \leq \frac{1}{s + p/n} = \frac{n}{p + n(1-p)} \leq n, \tag{1.19}$$

where s is the fraction of execution time spent in serial portions of the codes and p is the fraction of execution time spent in parallelizable portions of the codes, and $s+p = 1$. Amdahl's Law expressed by (1.19) is based on the assumption that there are only two different types of processing modes in the code; those which can be carried out in parallel on n processors and those which can only be carried out on a single processor. The execution time of a code run in sequential mode can be monitored by using a profiling utility which keeps track of the time spent within each subroutine or function. In the UNIX operating system this utility is called *gprof* (or *lprof*) [30]. By reading the output of *gprof*, we can get a rough idea of the profile of the code. To get an accurate profile, we need to use the FORTRAN library function ETIME, which returns elapsed user time and system time.

In our implementation of the parallel collocation algorithm, we are only interested in parallel execution of the operations which can be carried out independently. Such operations can be executed in parallel in any order. Our parallel implementation will be done on a eight processor system Alliant/FX2800. To execute independent operations in parallel on the Alliant/FX2800, we must first put these operations into

a subroutine. The subroutine calls are executed in parallel within a "parallel" DO loop, identified by means of a compiler directive. We provide an example to illustrate this.

Let us suppose we want to add two vectors $\vec{a}$ and $\vec{b}$. On an Alliant/FX2800, a parallel add is performed as follows,

```
      ...
CVD$  CNCALL
      DO 10 I=1,N
         CALL SUMAB(A(I),B(I))
   10 CONTINUE
      ...
      STOP
      END


      RECURSIVE SUBROUTINE SUMAB(A,B)
      REAL A, B
      A = A+B
      RETURN
      END
```

The compiler directive CVD$ CNCALL makes the DO loop run in parallel. To run the subroutine SUMAB in parallel, the key word RECURSIVE must be placed before the word SUBROUTINE and the routine must be compiled with the *-recursive* switch.

Chapter 2

Elliptic Case

Our ultimate goal is to develop software for linear parabolic differential equations in two space variables subject to linear boundary conditions. The spatial dependence of the parabolic equation is expressible in terms of an elliptic operator. The spatial discretization of this operator is a significant step in the numerical solution of the parabolic problem. Therefore in this chapter we look at the construction of a code to solve linear elliptic boundary value problems. It turns out that much of this work will be incorporated into the parabolic code. Concentrating on the spatial discretization allows us, for the moment, to ignore complications arising from the time variable.

2.1 Introduction

We consider here linear boundary value problems of the form,

$$(L_1 + L_2)u = f(x_1, x_2), \quad (x_1, x_2) \in \Omega = [0, 1] \times [0, 1], \tag{2.1}$$

$$u(x_1, x_2) = 0, \quad (x_1, x_2) \in \partial\Omega, \tag{2.2}$$

where

$$L_1 = -a_1(x_1)\frac{\partial^2}{\partial x_1^2} + c_1(x_1), \quad a_1(x_1) > 0 \text{ in } [0, 1], \tag{2.3}$$

and

$$L_2 = -a_2(x_2)\frac{\partial^2}{\partial x_2^2} + b_2(x_2)\frac{\partial}{\partial x_2} + c_2(x_2), \quad a_2(x_2) > 0 \text{ in } [0,1]. \tag{2.4}$$

One reason for considering problem (2.1) is that a broad class of boundary value problems is in the form of (2.1), such as elliptic boundary value problems in polar coordinates with x_1 taking the role of angular coordinate and x_2 being the radius coordinate. The absence of the $\frac{\partial}{\partial x_1}$ term ensures that we will get a symmetric-definite generalized eigenvalue problem in the matrix decomposition step, as we will show later in this chapter. Homogeneous Dirichlet boundary conditions (2.2) are considered here for simplicity. We will later extend the discussion to nonhomogenous Neumann, Dirichlet and mixed boundary conditions.

In order to describe the numerical procedure, we need some notation. This notation is used in [11] and [24].

For $i = 1, 2$, let N_i be a positive integer and let $\pi_i = \{x_i^{(j)}\}_{j=0}^{N_i}$ denote a partition of $[0, 1]$ such that,

$$0 = x_i^{(0)} < x_i^{(1)} < \cdots < x_i^{(N_i)} = 1\,,$$

and let $I_i^{(j)}$ denote the subinterval $[x_i^{(j-1)}, x_i^{(j)}]$, $j = 1, 2, \cdots, N_i$.

Let $r_i \geq 3$ and let $\mathcal{M}_i(r_i, \pi_i)$ be the space of piecewise polynomial functions defined by,

$$\mathcal{M}_i(r_i, \pi_i) = \{v \in C^1[0, 1] : v|_{[x_i^{(j-1)}, x_i^{(j)}]} \in P_{r_i},\ j = 1, \ldots N_i,\ v(0) = v(1) = 0\}, \tag{2.5}$$

where P_{r_i} denote the set of all polynomials of degree $\leq r_i$, $i = 1, 2$. The dimension of $\mathcal{M}_i(r_i, \pi_i)$ is $M_i = N_i(r_i - 1)$. Let $\{\sigma_i^{(k)}\}_{k=1}^{r_i-1}$ and $\{w_i^{(k)}\}_{k=1}^{r_i-1}$ be the nodes and weights of the $(r_i - 1)$ point Gauss-Legendre quadrature rule on $[0, 1]$, see [18]. The Gaussian points on $I_i^{(j)}$ are defined by,

$$\xi_i^{((j-1)(r_i-1)+k)} = x_i^{(j-1)} + h_i^{(j)}\sigma_i^{(k)}, \quad j = 1, \ldots, N_i, \quad k = 1, \ldots, r_i - 1,$$

where $h_i^{(j)} = x_i^{(j)} - x_i^{(j-1)}$, $i = 1, 2$.

2.2 Orthogonal Spline Collocation

Let $\{\phi_{n_i}^{(i)}\}_{n_i=1}^{M_i}$ be a set of basis function for $\mathcal{M}_i(r_i, \pi_i)$, where $M_i = N_i(r_i - 1), i = 1, 2$. We denote the space spanned on products of functions, one from $\mathcal{M}_1(r_1, \pi_1)$ and one from $\mathcal{M}_2(r_2, \pi_2)$, by $\mathcal{M}_1(r_1, \pi_1) \otimes \mathcal{M}_2(r_2, \pi_2)$. That is

$$\mathcal{M}_1(r_1, \pi_1) \otimes \mathcal{M}_2(r_2, \pi_2) = \text{the linear span of } \{\phi_{n_1}^{(1)}\phi_{n_2}^{(2)}\},$$

where $n_1 = 1, \cdots, M_1$ and $n_2 = 1, \cdots, M_2$. Suppose that $U \in \mathcal{M}_1(r_1, \pi_1) \otimes \mathcal{M}_2(r_2, \pi_2)$ is given by

$$U(x_1, x_2) = \sum_{n_1=1}^{M_1} \sum_{n_2=1}^{M_2} u_{n_1,n_2} \phi_{n_1}^{(1)}(x_1)\phi_{n_2}^{(2)}(x_2). \tag{2.6}$$

The orthogonal spline collocation approximate solution is obtained by requiring U given by (2.6) to satisfy (2.1) at the Gaussian points,

$$(L_1 + L_2)U(\xi_1^{(m_1)}, \xi_2^{(m_2)}) = f(\xi_1^{(m_1)}, \xi_2^{(m_2)}), \quad m_i = 1, \ldots, M_i, \tag{2.7}$$

where $M_i = N_i(r_i - 1), i = 1, 2$.

Let

$$\vec{u} = [u_{1,1}, u_{1,2}, \ldots, u_{1,M_2}, \ldots, u_{M_1,1}, \ldots, u_{M_1,M_2}]^T, \tag{2.8}$$

and

$$\vec{f} = [f_{1,1}, f_{1,2}, \ldots, f_{1,M_2}, \ldots, f_{M_1,1}, \ldots, f_{M_1,M_2}]^T, \tag{2.9}$$

where $f_{m_1,m_2} = f(\xi_1^{(m_1)}, \xi_2^{(m_2)})$. Then (2.7) can be written as,

$$(A_1 \otimes B_2 + B_1 \otimes A_2)\vec{u} = \vec{f}, \tag{2.10}$$

where

$$A_i = (a_{mn}^{(i)})_{m,n=1}^{M_i}, \quad a_{mn}^{(i)} = L_i\phi_n^{(i)}(\xi_i^{(m)}), \tag{2.11}$$

$$B_i = (b_{mn}^{(i)})_{m,n=1}^{M_i}, \quad b_{mn}^{(i)} = \phi_n^{(i)}(\xi_i^{(m)}), \quad i = 1, 2. \tag{2.12}$$

Matrices A_i, B_i, $i = 1, 2$, can be easily shown to have the almost block diagonal structure described in Chapter 1. See [10] for existence and uniqueness of U and its

convergence rate in the H^1-norm for some special cases of separable boundary value problems.

We will need the following lemmas. Lemma 2.1 here is Lemma 3.1 in [11].

Lemma 2.1 *Let* $W = diag(h_1^{(1)}w_1^{(1)}, \cdots, h_1^{(1)}w_1^{(r_1-1)}, \cdots, h_1^{(N_1)}w_1^{(1)}, \cdots, h_1^{(N_1)}w_1^{(r_1-1)})$, $D = diag(1/a_1(\xi_1^{(1)}), \cdots, 1/a_1(\xi_1^{(M_1)}))$ *and let* $\{\phi_{n_1}^{(1)}\}_{n_1=1}^{M_1}$ *be a set of basis functions for* $\mathcal{M}_1(r_1, \pi)$. *Let* A_1 *and* B_1 *be given by (2.11) and (2.12). Let* $F_1 = B_1^T W D B_1$, $G_1 = B_1^T W D A_1$. *Then* F_1 *is a symmetric positive definite matrix and* G_1 *is a symmetric matrix.*

Another lemma which will be useful, and which we will refer to here as Lemma 2.2, is Corollary 8.7.2 in [29].

Lemma 2.2 *If* G *is a symmetric matrix and* F *is a symmetric positive definite matrix, then there exists a nonsingular matrix* Z *such that*

$$Z^T G Z = \Lambda, \quad Z^T F Z = I, \tag{2.13}$$

where Λ *is a diagonal matrix and* I *is the identity matrix.*

It follows from Lemma 2.1 and Lemma 2.2 that there exist a diagonal matrix Λ and a nonsingular matrix Z such that,

$$Z^T B_1^T W D A_1 Z = Z^T G_1 Z = \Lambda, \tag{2.14}$$

and

$$Z^T B_1^T W D B_1 Z = Z^T F_1 Z = I_1, \tag{2.15}$$

where I_1 is the identity matrix of order M_1. As a result we have,

$$\begin{aligned}
&(Z^T B_1^T W D \otimes I_2)(A_1 \otimes B_2 + B_1 \otimes A_2)(Z \otimes I_2) \\
&= (Z^T B_1^T W D A_1 Z) \otimes B_2 + (Z^T B_1^T W D B_1 Z) \otimes A_2 \\
&= (Z^T G_1 Z) \otimes B_2 + (Z^T F_1 Z) \otimes A_2 \\
&= \Lambda \otimes B_2 + I_1 \otimes A_2,
\end{aligned} \tag{2.16}$$

where $\Lambda = diag(\lambda_i)_{i=1}^{M_1}$ and I_2 is the identity matrix of order M_2.

The matrix decomposition approach used here simplifies the linear system (2.10) to a linear system with the matrix $\Lambda \otimes B_2 + I_1 \otimes A_2$. As we will see, the algorithm given in next section is based on this matrix decomposition.

2.3 The Algorithm and Implementation

The algorithm we implement is identical to Algorithm I in [11].

Given A_i, B_i, $i = 1, 2$, as (2.11) and (2.12), W and D as in Lemma 2.1, $F_1 = B_1^T W D B_1$ and $G_1 = B_1^T W D A_1$.

Step 1 Determine Λ and Z satisfying (2.14) and (2.15).

Step 2 Compute $\vec{g} = (Z^T B_1^T W D \otimes I_2)\vec{f}$.

Step 3 Solve $(\Lambda \otimes B_2 + I_1 \otimes A_2)\vec{v} = \vec{g}$.

Step 4 Compute $\vec{u} = (Z \otimes I_2)\vec{v}$.

Algorithm 2.1

The parallel features of the above algorithm have been discussed in [11]. We will give some implementation details here.

To apply the algorithm one needs to set up the collocation matrices A_i and B_i, $i = 1, 2$. To do that we first need the nodes and weights of the $(r-1)$ point Gauss-Legendre quadrature rule on [0, 1]. Rather than storing these as constants for some range of values of r, these are obtained by calculating the eigenvalues of a symmetric tridiagonal matrix of order $r \times r$ [18]. The amount of work required is negligible and this improves the portability of our program. The eigenvalue problem for the symmetric tridiagonal matrix is solved by the EISPACK routines, IMTQL1 and IMTQL2 [26]. Next, we need to calculate the matrices A_i, B_i, $i = 1, 2$. Since we choose B-spline functions as the basis functions for $\mathcal{M}_1$ and $\mathcal{M}_2$, we could use the algorithms

for calculating B-splines given by de Boor [14], and implemented in a FORTRAN package in [13]. Our use of the B-splines does not require the full generality of de Boor's package, and as noted by Ascher and Russell in [6], improvements in efficiency are possible. Ascher and Russell have implemented an algorithm to do so, in [6], and we use this implementation in our codes.

In Step 1, to find Λ and Z satisfying (2.14) and (2.15), we need to construct the matrices $F_1 = B_1^T W D B_1$ and $G_1 = B_1^T W D A_1$. The matrices B_1, A_1 have the almost block diagonal structure described in Chapter 1. The matrices $B_1^T W D B_1$ and $B_1^T W D A_1$ have a more complex block structure, but are symmetric. Since there is no available software for computing eigenvalues of matrices of the form $B_1^T W D B_1$ which takes advantage of the block structure, we use band symmetric software. The band matrix chosen to represent $B_1^T W D B_1$, for example, may be found in one of two ways:

(i) represent B_1 as a band matrix and form $B_1^T W D B_1$;

(ii) form the matrix $B_1^T W D B_1$ keeping B_1 as an almost block diagonal and then representing the result as a band matrix.

We now discuss these two approaches. Fig. 2.1 shows the structure of the matrix B_1 for $N_1 = 4$ and $k = 4$. The matrix A_1 has the same structure as B_1. Here we use N_1 to denote the number of partition along the x_1 direction and k to denote the number of collocation points in each subinterval of the partition. V_1 is a $k \times (k-1)$ matrix; R_i, $i = 1, \ldots, , N_1 - 1$, are $k \times 2$ matrices, R_{N_1} is a $k \times (k-1)$ matrix, V_i, $i = 2, \ldots, N_1 - 1$, are $k \times (k-2)$ matrix, W_i, $i = 2, \ldots, N_1$, are $k \times 2$ matrices. If we take the matrix B_1 as a banded matrix, then the bandwidth for B_1 is $2k+2$, and the bandwidth for the matrix $B_1^T W D B_1$ will be $4k+1$. If instead we take B_1 as an almost block diagonal matrix, and recall that W and D are diagonal, we see that $B_1^T W D B_1$ will have the same form as $B_1^T B_1$, shown in Fig. 2.2. It is easy to see the bandwidth for the matrix in Fig. 2.2 is $2k+2$. Thus there are $2k-1$ words of storage saved per row if we choose the data structure for the matrix B_1 to be almost

block diagonal instead of taking B_1 as banded. The same discussion applies to A_1 and the formation of $B_1^T W D A_1$. While the computation costs for both (i) and (ii) are $O(N_1 k^3)$, it is clear that (ii) is more efficient in time and storage than (i). Compared to the total cost of the Algorithm 2.1 these costs are of lower order.

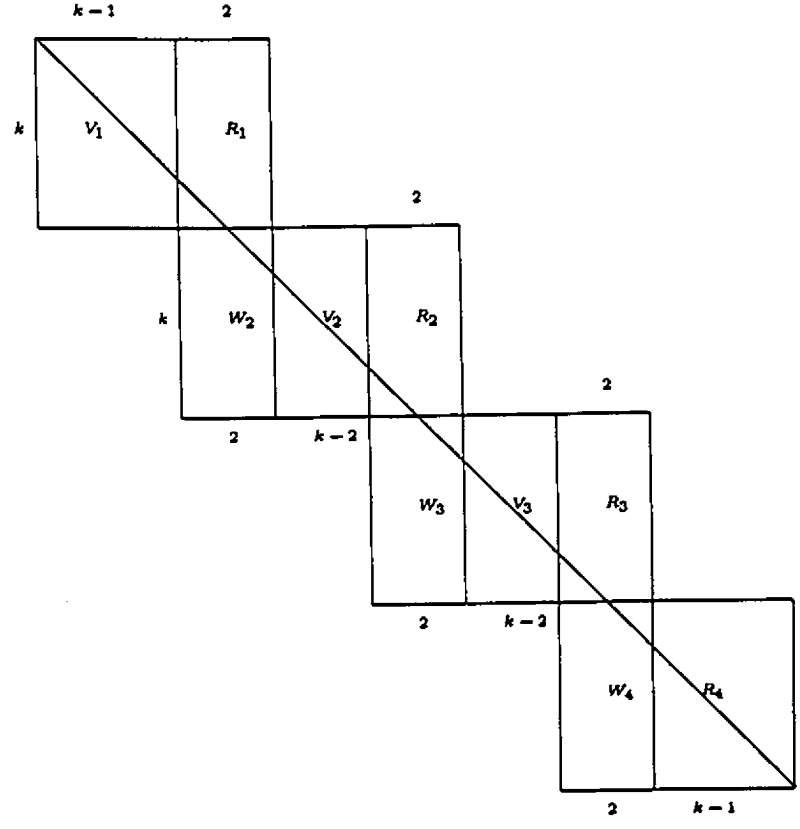

Figure 2.1: The ABD structure of the collocation matrix B_1.

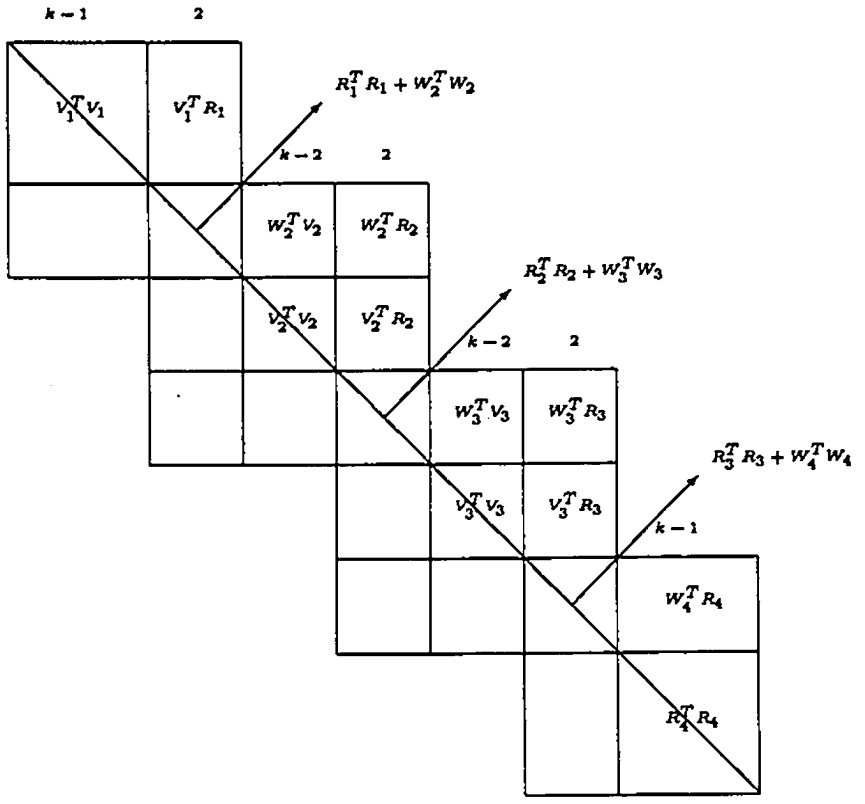

Figure 2.2: The structure of the matrix $B_1^T B_1$.

It is easy to show that finding matrices Λ and Z such that,

$$Z^T G_1 Z = \Lambda, \quad Z^T F_1 Z = I_1,$$

is equivalent to solving a symmetric generalized eigenvalue problem of the form,

$$G_1 Z = F_1 Z \Lambda, \quad Z^T F_1 Z = I_1. \tag{2.17}$$

This follows from multiplying both sides of the first equation in (2.17) by Z^T and then using the second equation. To solve the first equation in (2.17), we use the NAG [49] routine F02FHF. This routine computes the eigenvalues of the generalized banded symmetric eigenvalue problem, using the method of Crawford [17], which takes advantage of the banded structure of the matrix. Crawford's method transforms the original problem to a standard banded symmetric eigenvalue problem. After the eigenvalues have been found, the eigenvectors corresponding to the eigenvalues can be found by independent calls to another NAG routine F02SDF, which uses an inverse iteration algorithm to obtain each eigenvector of (2.17). Peters and Wilkinson [58] give a detailed discussion of the procedure for finding the eigenvector corresponding to a given eigenvalue. Since the eigenvalue is accurate, one iteration will usually give a small residual and thus a good approximation for the eigenvector. The calculations for each eigenvector may be carried out in parallel. The matrix of computed eigenvectors, which we denote by Z_1, does not necessarily satisfy the normalization equation. However $Z_1^T F_1 Z_1$ will be a diagonal matrix, say $S = diag(d_i)_{i=1}^{M_1}$. Since F_1 is a positive definite matrix by Lemma 2.1, $d_i > 0$, $i = 1, \ldots, M_1$, and we can take $Z = Z_1 S^{-\frac{1}{2}}$, which gives $Z^T F_1 Z = I_1$. (The matrix decomposition algorithm could be modified for $Z^T F_1 Z = S \neq I_1$ to avoid computing $S^{\frac{1}{2}}$. Some minor savings will result.) The calculation of Z from Z_1 is simply a row scaling of Z_1 and can also be done in parallel.

In Step 2, we have

$$\vec{g} = (Z B_1^T W D \otimes I_2)\vec{f} = P^T (I_2 \otimes Z B_1^T W D) P \vec{f}, \tag{2.18}$$

where P is the permutation matrix such that

$$P\vec{f} = [f_{1,1}, f_{2,1}, \ldots, f_{M_2,1}, \ldots, f_{1,M_2}, \ldots, f_{M_1,M_2}]^T. \tag{2.19}$$

Then,

$$I_2 \otimes (ZB_1^T W D) P\vec{f} = I_2 \otimes (ZB_1^T W D)\vec{y} = \begin{bmatrix} ZB_1^T W D\vec{y}_1 & & \\ & \ddots & \\ & & ZB_1^T W D\vec{y}_{M_2} \end{bmatrix},$$

where $\vec{y} = P\vec{f}$ and $\vec{y} = (\vec{y}_1^T, \cdots, \vec{y}_{M_2}^T)^T$. The calculations within the tensor product are thus decoupled and can be done in parallel.

In Step 3, the structure of $(\Lambda \otimes B_2 + I_1 \otimes A_2)$ is such that it can be broken into independent subsystems, each of which is an ABD system. Solving each subsystem in parallel would involve another level of parallel granularity. Several authors have proposed parallel algorithms for solving almost block diagonal systems (see [39], [57], [72], [73]). While several experimental codes based on these algorithms are being developed, no robust parallel ABD solver is currently available. Hence we solve each of the subsystems using the sequential solver ARCECO [19], [20], which is a FORTRAN package for solving almost block diagonal linear systems by modified alternate row and column elimination. Any new parallel ABD solver could be easily incorporated into our software.

In Step 4, we have

$$\vec{u} = (Z \otimes I_2)\vec{v} = P^T(I_2 \otimes Z)P\vec{v}, \tag{2.20}$$

where P is the permutation introduced in (2.18). As in Step 2, this decouples the calculations so that they are suitable for parallel computation.

In our implementation of algorithm 2.1, level 1 BLAS, [44], [45], and level 2 BLAS, [22], [23], routines have been employed. Level 1 BLAS routines are basic linear algebra programs targeted at vector-vector operations and Level 2 BLAS routines are targeted at matrix-vector operations.

In the next section we report on the numerical testing of ELLDCM which is our implementation of the algorithm described here.

2.4 Performance Analysis and Numerical Experiments

Our code ELLDCM for elliptic matrix decomposition, is written with parallelism in mind. To enable parallelization, we need to avoid data dependency in loops, since parallel computation may be prevented in the presence of data dependency variables. Such a variable is one which is stored in one statement and subsequently appears on the right hand side of another statement. For example, for the following code

```
   ...
   X = 0.0
   DO 10 I=1,N
      A(I) = A(I) + B(I)
      X = X + B(I) * B(I)
10 CONTINUE
      X = SQRT(X)
   ...
```

the variable X prevents the loop from being run in parallel, since the value X depends on the X in the the previous loop iteration. Hence X is a data dependency variable. However if we break the loop into two loops as follows:

```
   ...
   X = 0.0
   DO 10 I=1,N
      X = X + B(I) * B(I)
10 CONTINUE
   X = SQRT(X)
   DO 20 I=1,N
      A(I) = A(I) + B(I)
20 CONTINUE
   ...
```

then in the second loop, each iteration is independent of the others and this loop may then be run in parallel.

In this section we demonstrate results obtained by ELLDCM on the selection of problems shown in Table 2.1. All test problems in Table 2.1 are in the form of (2.1)-(2.4). The right hand function $f(x_1, x_2)$ is computed using its final form obtained by simplifying $(L_1 + L_2)u$, where u is the true solution given. With the UNIX utility

command *gprof*, we obtained a rough execution time profile in sequential mode, which is an account of the amount of CPU execution time spent in each subroutine. The output of *gprof* tells us which parts of the program are taking most of the execution time. However, in order to get more accurate times for the profiling we report in this section, we use the more accurate FORTRAN library function, ETIME. We first give results on profiling to show how parallelism improves the performance. Then we give some convergence results and carry out comparisons with Kaufman and Warner's code, SERRG2 [41].

(2.1)	$L_1 = -(1+x_1^3)\frac{\partial^2}{\partial x_1^2} + x_1,$
	$L_2 = -(1+x_2^2)\frac{\partial^2}{\partial x_2^2}) - x_2\frac{\partial}{\partial x_2} + x_2^2,$
	True solution: $u = \sin(\pi x_1)\sin(\pi x_2)$
(2.2)	$L_1 = -\frac{1+x_1}{1+x_1^2}\frac{\partial^2}{\partial x_1^2} + \frac{1}{1+x_1},$
	$L_2 = -\frac{1}{1+x_2^2}\frac{\partial^2}{\partial x_2^2} + \frac{x_2}{1+x_2}\frac{\partial}{\partial x_2} - \frac{3}{1+x_2},$
	True solution: $u = 4\frac{\sin(\pi x_1)\sin(\pi x_2)}{1+x_1^2+x_2^2}$
(2.3)	$L_1 = -e^{x_1}(x_1\sqrt{x_1}+1)\frac{\partial^2}{\partial x_1^2} + e^{x_1},$
	$L_2 = -(e^{x_2}+1)\frac{\partial^2}{\partial x_2^2} + e^{x_2}\frac{\partial}{\partial x_2} + 1,$
	True solution: $u = \sin(\pi x_1)\sin(\pi x_2)$

Table 2.1: Test problems for the elliptic solver.

2.4.1 Time Profiling and Speedup

We solve the test problems in Table 2.1 with M = 100 and $k = 2$, where M is the number of subintervals and k is the number of collocation points in each subinterval. Here we choose the same partition and the same number of collocation points in

the x_1 and x_2 directions. The profiling was carried out on an Alliant/FX2800. We are mainly interested in the routines which can be executed through a sequence of independent calls. The prospects for efficient parallelization are good since these routines perform about 90% of the work when the code is run in sequential mode. These routines, ZTATV, ZV, EIGENV, ARCE and FUN, are described as follows:

- ZTATV is the subroutine to calculate $Z^T A^T \vec{v}$ where Z is a full matrix, A^T is an almost block diagonal matrix, $\vec{v}$ is a vector. This routine is needed to carry out Step 2 in Alg. 2.1.

- ZV is the subroutine to calculate $Z\vec{v}$ where Z is a full matrix and $\vec{v}$ is a vector. This routine is needed to carry out Step 4 in Alg. 2.1.

- EIGENV is a routine to find the eigenvector corresponding to a given real eigenvalue for the generalized problem $A\vec{x} = \lambda B\vec{x}$ or for the standard eigenvalue problem $A\vec{x} = \lambda\vec{x}$, where A and B are real band matrices. EIGENV calls the NAG routine F02SDF [49], mentioned in the previous section.

- ARCE is the subroutine to solve an almost block diagonal linear system, this routine calls ARCECO and it is used to carry out Step 3 in Alg. 2.1.

- FUN is the subroutine to evaluate the right hand side function of the differential equation.

In Table 2.2 we present the running time of each routine for solving the test problems in Table 2.1. Time is given in seconds. We use N_P to denote the number of processors, CPU_T to denote CPU time and % to denote the percentage of the running time which the routine takes. We also provide the overall running time for ELLDCM.

N_P	Routine	Problem 2.1		Problem 2.2		Problem 2.3	
		CPU_T	%	CPU_T	%	CPU_T	%
1	EIGENV	13.68	29.00	13.62	24.74	13.59	28.30
	ZTATV	10.98	23.27	11.50	20.90	11.48	23.90
	ZV	12.35	26.17	12.34	22.41	12.55	26.13
	ARCE	3.79	8.03	3.62	6.58	3.77	7.86
	FUN	2.83	6.00	10.4	18.98	2.76	5.75
	ELLDCM	47.18		55.06		48.03	
2	EIGENV	6.85	25.73	6.89	22.73	6.61	26.06
	ZTATV	5.50	20.67	5.55	18.34	5.29	20.85
	ZV	6.29	23.65	6.21	20.51	6.09	24.02
	ARCE	1.95	7.35	1.86	6.16	1.86	7.35
	FUN	1.79	6.75	5.70	18.81	1.66	6.56
	ELLDCM	26.62		30.30		25.38	
4	EIGENV	3.37	23.12	3.54	20.55	3.38	22.56
	ZTATV	2.77	19.02	2.72	15.77	2.72	18.21
	ZV	2.93	20.12	3.01	17.45	3.10	20.70
	ARCE	0.87	6.02	0.921	5.33	0.93	6.23
	FUN	0.88	6.04	3.08	1788	0.82	5.49
	ELLDCM	14.57		17.26		14.98	
8	EIGENV	1.76	17.24	1.79	16.37	1.76	17.48
	ZTATV	1.57	15.32	1.53	13.98	1.72	17.06
	ZV	1.78	17.43	1.72	15.69	1.72	17.10
	ARCE	0.49	4.80	0.482	4.38	0.48	4.81
	FUN	0.48	4.74	1.50	13.66	0.42	4.20
	ELLDCM	10.24		10.98		10.09	

Table 2.2: Profiling for test problems 2.1-2.3.

Table 2.2 shows that the running time of the routines which can be built into a sequence of independent calls decreases in a linear fashion as the number of processor increases. In order to determine the maximum speedup, we need to calculate the the percentage of time spent in the parallelizable code when run on a single processor. This fraction p equals the sum of the times spent on EIGENV, ZTATV, ZV, ARCE and FUN divided by the total time. In Table 2.3, we present the actual speedup we obtained and the maximum speedup we can expect. The maximum speedup is calculated by Amdahl's law (1.19) with $n = 8$ and the actual speedup is computed by (1.18), given in section 1.2.4.

N_P	Problem 2.1	Problem 2.2	Problem 2.3
2	1.77	1.81	1.89
4	3.23	3.19	3.20
8	4.60	5.01	4.76
Max. Speedup	5.24	5.53	5.11

Table 2.3: Speedup of the test problems 2.1-2.3 on an Alliant/FX2800.

2.4.2 Convergence

In the following, we describe some numerical results which show the convergence properties of the orthogonal spline collocation algorithm on test problems 2.1-2.3 given in Table 2.1 for $k = 2$ and $k = 3$, where k is the number of collocation points in each subinterval. In Figures 2.3-2.5, we plot on a log-log scale the L_∞ errors and maximum errors at the mesh points of the approximate solution obtained using $h_i^{(j)} \equiv h$, $j = 1, \cdots, N_i$, $i = 1, 2$, for $h = 1/4$, $1/8$, $1/16$, $1/32$, $1/64$. The L_∞ errors are computed on a 100×100 uniform grid. Recall that the rate of convergence can be estimated by (1.13). The slopes of the lines in the log-log plot approximately equal the rates of convergence. The degradations of the slopes for $k = 3$ and $-\log(h) = 5$

and 6 cases are because the approximation errors are so small that roundoff error becomes significant.

From an examination of the figures, we can see that the slopes of the solid lines corresponding to $k = 2$ and $k = 3$ are approximately 4 and 5 while the slopes of the dashed lines corresponding to $k = 2$ and $k = 3$ are approximately 4 and 6 respectively. Hence the convergence rates of the L_∞ errors, corresponding to $k = 2$ and $k = 3$, are 4 and 5 respectively and the convergence rates of the maximum mesh point errors, corresponding to $k = 2$ and $k = 3$, are 4 and 6 respectively. The convergence rate of the maximum error at mesh points corresponding to $k = 3$ demonstrates the superconvergence. The data for Figure 2.3 are given in Table 2.4, where we use E_u to denote the L_∞ error, E_m to denote the maximum error at mesh points, and R_u and R_m to denote values computed by (1.13) for consecutive step sizes. For the $k = 3$ and $-\log(h) = 5$ and 6 cases the approximation error is so small in double precision, that roundoff error becomes significant.

	k=2				k=3			
$-\log(h)$	$-\log(E_u)$	R_u	$-\log(E_m)$	R_m	$-\log(E_u)$	R_u	$-\log(E_m)$	R_m
2	12.36		14.32		18.51		24.99	
3	16.36	4.00	18.25	3.93	23.49	4.98	31.06	6.07
4	20.35	3.99	22.24	3.99	28.52	5.03	37.06	6.00
5	24.35	4.00	26.23	3.99	33.50	4.98	43.66	6.60
6	28.36	4.01	30.23	4.00	38.48	4.98	43.88	0.22

Table 2.4: Data for plotting Figure 2.3.

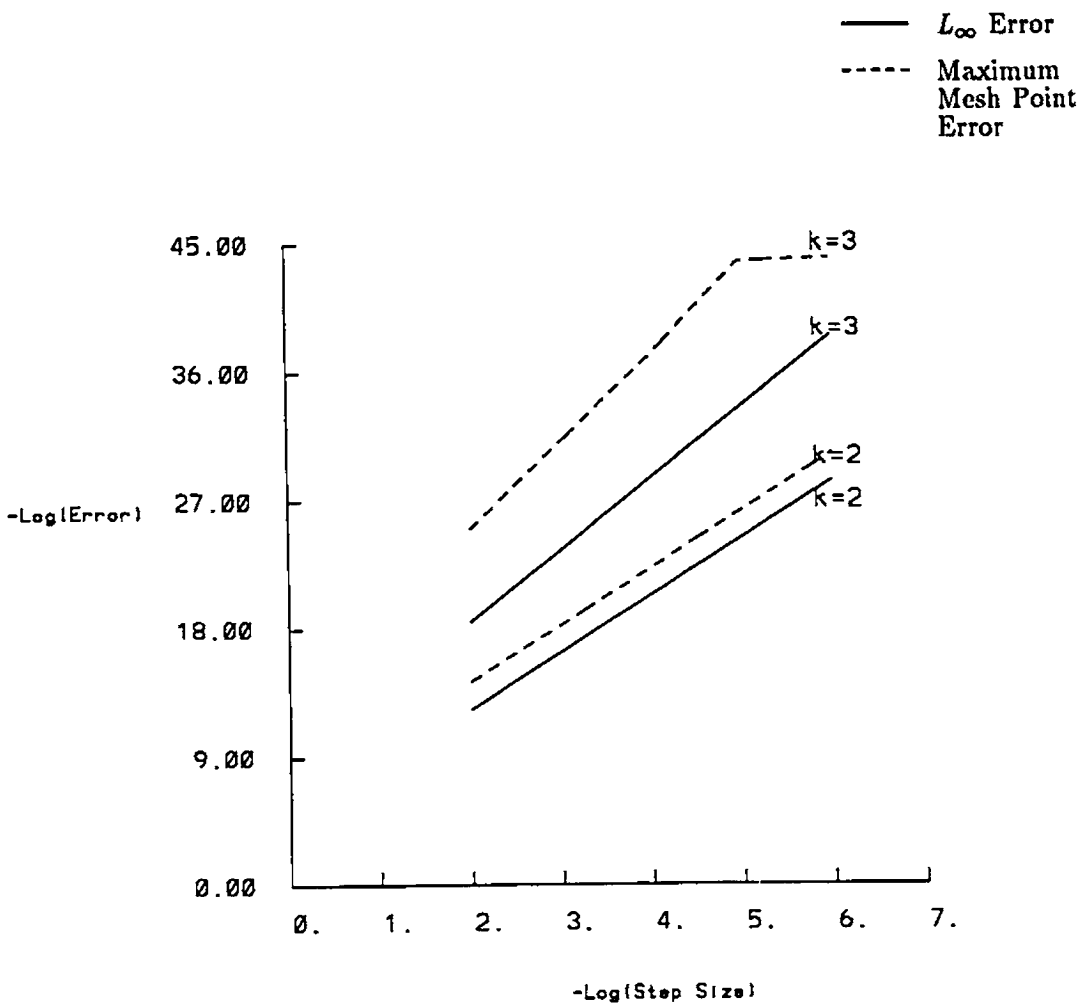

Figure 2.3: Convergence results for test problem 2.1 (k=2, 3).

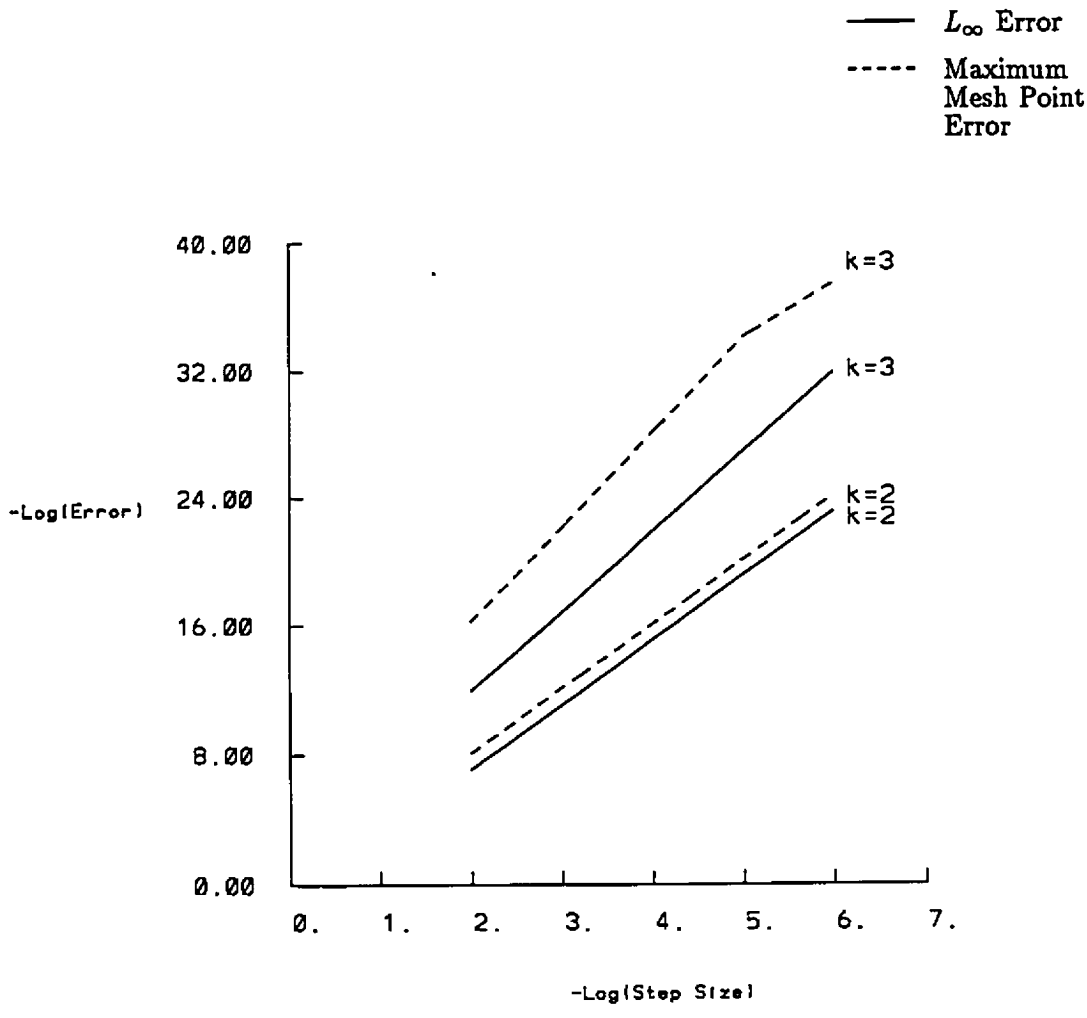

Figure 2.4: Convergence results for test problem 2.2 (k=2, 3).

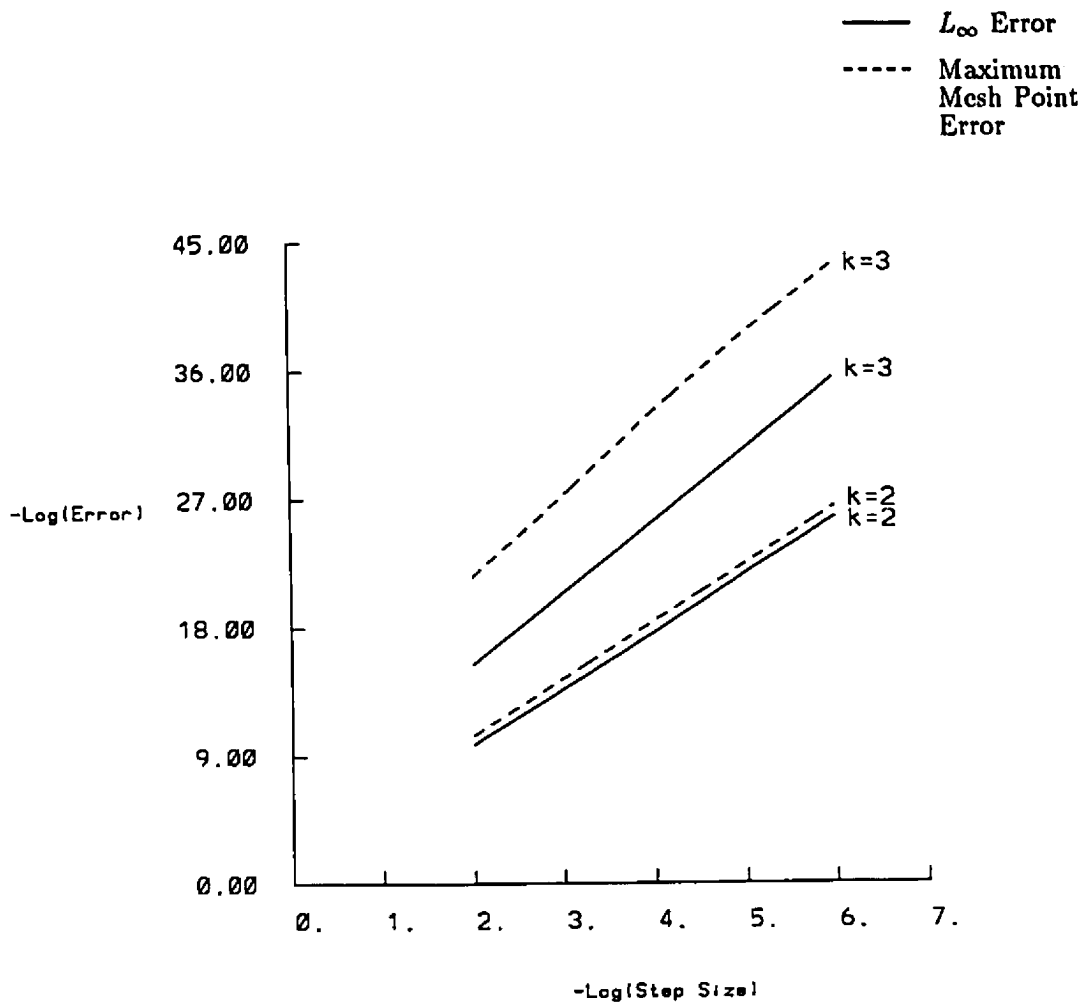

Figure 2.5: Convergence results for test problem 2.3 (k=2, 3).

2.4.3 Comparison with SERRG2

Over the past 30 years, much research has been done on the numerical approximation of solutions of partial differential equations of elliptic type. Prior to 1980, as mentioned in [50], the general packages [1], [38], [63], [64], [67] and [68] were already available. More recent work in this area includes, e.g. [37] and [41].

We have compared our program, ELLDCM, with the recent code SERRG2 [41], by Kaufman and Warner, since it is the closest competitor. The program SERRG2 solves separable elliptic equations on a rectangular region, using a matrix decomposition technique to solve the linear system arising from a Rayleigh-Ritz-Galerkin approximation with tensor product B-spline basis functions. The elliptic problems which can be handled by SERRG2 are of the form:

$$(L_1 + L_2)u = f(x_1, x_2), \tag{2.21}$$

where

$$L_1 = -\frac{\partial}{\partial x_1}(p_1(x_1)\frac{\partial}{\partial x_1}) + r(x_1)\frac{\partial}{\partial x_1} + q_1(x_1), \quad p_1(x_1) > 0, \tag{2.22}$$

$$L_2 = -\frac{\partial}{\partial x_2}(p_2(x_2)\frac{\partial}{\partial x_2}) + q_2(x_2), \quad p_2(x_2) > 0. \tag{2.23}$$

Dirichlet, mixed and periodic boundary conditions are permitted by SERRG2. Similar to our code ELLDCM, SERRG2 finds the coefficients of a tensor-product B-splines approximation to (2.21). As in ELLDCM, this involves solving a generalized eigenvalue problem, similar to (2.17). A set of linear equations in matrix tensor products is obtained, in which each element of the coefficient matrix and the right hand side of the linear system requires an integration. Compared to orthogonal spline collocation, this is a major drawback, since in orthogonal spline collocation each entry of the coefficient matrix is simply a function evaluation. After setting up the matrix, SERRG2 uses the algorithm SESLV described in [41] to solve the linear system in order to get the coefficients of the approximate tensor-product B-splines solution.

Following [41], we use

K - the order of the B-spline approximation, $K = k + 2$.

M - the number of subintervals.

We solve the following Poisson equation with both SERRG2 and ELLDCM

$$\left.\begin{aligned} &(L_1 + L_2)u = f(x_1, x_2), \\ &u(x_1, x_2) = 0, \quad (x_1, x_2) \in \partial\Omega, \end{aligned}\right\} \tag{2.24}$$

where

$$L_1 = -(x_1^2 + 1)\frac{\partial^2}{\partial x_1^2} + \sqrt{x_1} \text{ and } L_2 = -(e^{x_2} + 1)\frac{\partial^2}{\partial x_2^2} - e^{x_2}\frac{\partial}{\partial x_2} + 1,$$

and f is chosen such that the true solution of the differential equation is

$$u(x_1, x_2) = e^{x_1+x_2}\sin(\pi x_1)\sin(\pi x_2).$$

Although the general problem classes that can be handled by ELLDCM and SERRG2 are different, they do overlap to some extent. Equation (2.24) lies within the overlap and can be treated by both programs. In Table 2.5 we report the results. We use E_u to denote the L_∞ error and E_m to denote the maximum error at mesh points. We calculate the L_∞ error on a 100×100 uniform grid. Time reported here is in seconds and sequential mode. The computations were carried out on a Sparc 1000 workstation. From Table 2.5, we can see that for a fixed order of B-spline and number of subintervals SERRG2 achieves slightly better accuracy than ELLDCM does, but the execution time of SERRG2 is much more than that of ELLDCM. For example, for $K = 4$ and $M = 64$, the L_∞ error of SERRG2 is about a half of the L_∞ error of ELLDCM, but the execution time for SERRG2 is about 7 times the execution time of ELLDCM. Also we can see that for a fixed accuracy, say 10^{-6}, the orthogonal spline collocation requires less time than SERRG2. Finally we notice that the orthogonal spline collocation achieves higher order of convergence at the mesh points.

K	M	SERRG2		ELLDCM		
		E_u	TIME	E_u	E_m	TIME
4	4	$2.70 \cdot 10^{-3}$	0.05	$5.43 \cdot 10^{-3}$	$1.57 \cdot 10^{-3}$	0.03
4	8	$2.17 \cdot 10^{-4}$	0.21	$3.35 \cdot 10^{-4}$	$9.83 \cdot 10^{-5}$	0.08
4	16	$1.43 \cdot 10^{-5}$	1.01	$2.09 \cdot 10^{-5}$	$6.30 \cdot 10^{-6}$	0.27
4	32	$8.94 \cdot 10^{-7}$	5.68	$1.31 \cdot 10^{-6}$	$3.94 \cdot 10^{-7}$	1.17
4	64	$5.59 \cdot 10^{-8}$	34.72	$8.18 \cdot 10^{-8}$	$2.46 \cdot 10^{-8}$	4.81
5	4	$6.30 \cdot 10^{-5}$	0.13	$1.69 \cdot 10^{-4}$	$1.01 \cdot 10^{-6}$	0.03
5	8	$1.76 \cdot 10^{-6}$	0.57	$5.20 \cdot 10^{-6}$	$1.76 \cdot 10^{-8}$	0.16
5	16	$5.23 \cdot 10^{-8}$	3.33	$1.65 \cdot 10^{-7}$	$2.99 \cdot 10^{-10}$	0.62
5	32	$1.59 \cdot 10^{-9}$	17.78	$5.01 \cdot 10^{-9}$	$4.61 \cdot 10^{-12}$	2.71
5	64	$5.48 \cdot 10^{-11}$	108.33	$1.54 \cdot 10^{-10}$	$1.15 \cdot 10^{-12}$	13.50
6	4	$1.70 \cdot 10^{-6}$	0.24	$3.37 \cdot 10^{-6}$	$2.28 \cdot 10^{-8}$	0.09
6	8	$2.95 \cdot 10^{-8}$	1.24	$6.05 \cdot 10^{-8}$	$9.51 \cdot 10^{-11}$	0.31
6	16	$4.59 \cdot 10^{-10}$	6.67	$9.95 \cdot 10^{-10}$	$4.07 \cdot 10^{-13}$	1.24
6	32	$7.50 \cdot 10^{-12}$	38.02	$1.27 \cdot 10^{-11}$	$1.97 \cdot 10^{-12}$	5.77
6	64	$4.09 \cdot 10^{-12}$	253.95	$6.53 \cdot 10^{-12}$	$6.51 \cdot 10^{-12}$	27.84

Table 2.5: Errors and running time of SERRG2 and ELLDCM on Equation (2.24).

After studying the time profile, we realized that SERRG2 spends a large portion of the running time on solving the generalized eigenvalue problem. This is not surprising since SERRG2 solves the generalized eigenvalue problem by using the EISPACK routine REDUC to transform the generalized eigenvalue problem to a standard symmetric eigenvalue problem. In [41], it is stated that "it was not worth the effort to use a Crawford-like scheme as in Crawford [17] and Kaufman [40] to avoid forming a full standard eigenvalue problem." To investigate this, we modified SERRG2 to use

the NAG implementation of Crawford's algorithm to solve the generalized eigenvalue problem, instead of using REDUC. This has an effect on the running time for the following reason. For the generalized eigenvalue problem,

$$Gx = \lambda Fx,$$

where both G and F are $n \times n$ symmetric band matrix with bandwidth r, Kaufman-Warner's approach is to transform the problem to a standard eigenvalue problem by forming the Cholesky decomposition of F and then applying the QR algorithm to the standard eigenvalue problem by first reducing it to a tridiagonal matrix problem obtained by orthogonal similarity transformations. As indicated in [29], page 424, the cost of the QR algorithm is $9n^3$. The factor 9 is caused by the accumulation of the eigenvectors. The cost involved in first reducing the banded generalized eigenvalue problem to a standard eigenvalue problem are of lower order than the cost of a single QR step in the standard eigenvalue problem and we therefore ignore it. On the other hand if we use Crawford's algorithm and inverse iteration to solve $Gx = \lambda Fx$, the total costs are only $O(rn^2)$ operations.

The principal difference between Kaufman-Warner's algorithm and ours is in the use of Crawford's method for the generalized eigenvalue problem instead of reducing to a standard eigenvalue problem. The difference in cost is $O(9n^3) - O(rn^2)$, where r is considerably less than n. Since the $O(n^3)$ operations are the most time consuming operations, there is a substantial saving when Crawford's algorithm is employed. In SERRG2 matrix-vector multiplications arise during the computation of $(I \otimes Z)\vec{g}$, where Z is a full matrix and $\vec{g}$ is a vector. This step is analogous to Step 2 and Step 4 in our Algorithm 2.1. In an attempt to improve its efficiency, we modified the matrix-vector multiplication routine in SERRG2 by calling the level 2 BLAS routine DGEMV as we did in ELLDCM. We compare the modified version of SERRG2 with ELLDCM and in Table 2.6 we report the results. The errors are not repeated in Table 2.6 since they are identical with the errors reported in Table 2.5.

M	$K=4$	$K=5$	$K=6$
4	0.03	0.11	0.20
8	0.18	0.44	0.94
16	0.63	1.88	3.94
32	2.74	7.72	17.11
64	12.37	35.95	80.29

Table 2.6: The running time of the modified SERRG2 on Equation (2.24).

We see that the running time of the modified SERRG2 is substantially reduced by 1/3 for $M = 64$. We have profiled both codes and have found that the time spent on solving the generalized eigenvalue problem and on matrix-vector multiplication is close to that of ELLDCM. SERRG2 still has a longer running time since the finite element Galerkin method involves the use of quadratures to set up the matrix and right hand side. Also SERRG2 treats the linear systems as banded and does not take advantage of the almost block diagonal structure as ELLDCM does in Step 3.

Next we consider an example for which a variable mesh is required. We chose a problem that can be treated by both ELLDCM and SERRG2. The problem is

$$\left.\begin{aligned} &(L_1 + L_2)u = (\sqrt{x_1} + 2)\pi^2 \sin(\pi x_1)\sin(\pi x_2), \\ &u(x_1, x_2) = 0, \ \ (x_1, x_2) \in \partial\Omega, \end{aligned}\right\} \tag{2.25}$$

where

$$L_1 = -(\sqrt{x_1} + 1)\frac{\partial^2}{\partial x_1^2} \ \text{ and } \ L_2 = -\frac{\partial^2}{\partial x_2^2}.$$

The true solution of the above differential equation is $\sin(\pi x_1)\sin(\pi x_2)$.

To use SERRG2, we need to transform (2.25) so that it conforms to the general form of (2.21)-(2.23) as follows,

$$\left.\begin{aligned} &(L_1 + L_2)u = (\sqrt{x_1} + 2)\pi^2 \sin(\pi x_1)\sin(\pi x_2), \\ &u(x_1, x_2) = 0, \ \ (x_1, x_2) \in \partial\Omega, \end{aligned}\right\} \tag{2.26}$$

where

$$L_1 = -\frac{\partial}{\partial x_1}\left((1+\sqrt{x_1})\frac{\partial}{\partial x_1}\right) + \frac{1}{2\sqrt{x_1}}\frac{\partial}{\partial x_1} \text{ and } L_2 = -\frac{\partial^2}{\partial x_2^2}.$$

Since the function $1/\sqrt{x_1}$ is singular at $x_1 = 0$, a uniform mesh is not appropriate, and therefore we choose the graded mesh to be $(I/M)^3$, $I = 0, \cdots, M$, which causes the points to cluster near the singularity. We also use this graded mesh for our code ELLDCM. In Table 2.7 we report the results.

		SERRG2		ELLDCM	
K	M	E_u	TIME	E_u	TIME
4	4	$8.79 \cdot 10^{-3}$	0.04	$1.10 \cdot 10^{-3}$	0.01
4	8	$8.04 \cdot 10^{-4}$	0.11	$7.68 \cdot 10^{-5}$	0.06
4	16	$7.36 \cdot 10^{-5}$	0.53	$4.92 \cdot 10^{-6}$	0.23
4	32	$5.15 \cdot 10^{-6}$	3.40	$2.80 \cdot 10^{-7}$	0.77
4	64	$5.25 \cdot 10^{-7}$	23.50	$1.84 \cdot 10^{-8}$	3.78
5	4	$6.10 \cdot 10^{-4}$	0.07	$2.28 \cdot 10^{-5}$	0.04
5	8	$3.93 \cdot 10^{-5}$	0.31	$7.05 \cdot 10^{-7}$	0.13
5	16	$1.34 \cdot 10^{-6}$	1.70	$2.19 \cdot 10^{-8}$	0.52
5	32	$4.96 \cdot 10^{-8}$	11.07	$6.76 \cdot 10^{-10}$	1.99
5	64	$4.12 \cdot 10^{-7}$	78.00	$2.14 \cdot 10^{-11}$	10.15
6	4	$5.40 \cdot 10^{-5}$	0.14	$1.22 \cdot 10^{-6}$	0.07
6	8	$9.92 \cdot 10^{-7}$	0.67	$2.15 \cdot 10^{-8}$	0.24
6	16	$2.06 \cdot 10^{-8}$	3.99	$3.46 \cdot 10^{-10}$	0.99
6	32	$2.67 \cdot 10^{-8}$	25.73	$4.42 \cdot 10^{-12}$	4.25
6	64	$1.52 \cdot 10^{-7}$	185.54	$2.53 \cdot 10^{-12}$	23.96

Table 2.7: The L_∞ error and running time of SERRG2 and ELLDCM on Equation (2.26).

The singularity in the derivative of the coefficient function does not effect ELLDCM, but appears to give problems in the integrations in SERRG2. This time we see that ELLDCM obtains a solution with a much smaller error than SERRG2 for a fixed B-spline order and number of subintervals. Because of the singularity in the coefficient function, there is a degradation in the accuracy for SERRG2 as the step size decreases. Since the functions which ELLDCM needs for the evaluations have no singularity, the errors from ELLDCM are as usual. The errors using SERRG2, provided all integrals are done exactly, should also be as usual. However SERRG2 needs to approximate integrals which involve the coefficients of the PDE and the singularity at zero causes a degradation in the accuracy of the quadrature rules. See, for example, Davis and Rabinowitz for a discussion of error in Gaussian quadrature. The running time of SERRG2 here is less than the running time for solving (2.24) due to the simplicity of the coefficient functions. When using Kaufman-Warner's code for solving a equation with complicated coefficient functions, we may expect high costs for setting up the matrix and the right hand side vector.

In this section we have shown that our code, ELLDCM, has better performance than SERRG2 for certain test problems. Even after we modified SERRG2 to use a more efficient generalized eigenvalue problem solver, ELLDCM still takes less running time because the orthogonal spline collocation matrices take less time to setup than the matrices arising in SERRG2.

2.5 On More General Boundary Conditions

In section 2.1 and subsequent sections, we considered a family of elliptic PDEs with homogeneous Dirichlet boundary conditions,

$$u(x_1, x_2) = 0, \ (x_1, x_2) \in \partial\Omega \ \text{ where } \ \Omega = [0,1] \times [0,1].$$

In this section we generalize the boundary conditions as follows: for scalar constants α, β, γ, δ, we define the functional operators $V_{\alpha,\beta}$ and $H_{\gamma,\delta}$ as

$$V_{\alpha,\beta}\, u(x_1, x_2) \ \equiv \ \alpha u(x_1, x_2) - \beta u_{x_1}(x_1, x_2), \tag{2.27}$$

$$H_{\gamma,\delta}\, u(x_1,x_2) \equiv \gamma u(x_1,x_2) - \delta u_{x_2}(x_1,x_2), \tag{2.28}$$

where u_{x_i} denotes $\frac{\partial u}{\partial x_i}$, $i = 1,2$. We consider (2.1), (2.3), (2.4), subject to the following boundary conditions,

$$V_{\alpha_0,\beta_0} u(0,x_2) \equiv g_0(x_2), \tag{2.29}$$

$$V_{\alpha_1,\beta_1} u(1,x_2) \equiv g_1(x_2),\ x_2 \in [0,1]; \tag{2.30}$$

$$H_{\gamma_0,\delta_0} u(x_1,0) \equiv h_0(x_1), \tag{2.31}$$

$$H_{\gamma_1,\delta_1} u(x_1,1) \equiv h_1(x_1),\ x_1 \in [0,1]. \tag{2.32}$$

The boundary conditions (2.29)-(2.32) are assumed to be consistent. For example, in the case of non-homogeneous Dirichlet conditions, when $\beta_i = \delta_i = 0$, $i = 0,1$, at $x_1 = x_2 = 0$, (2.29), (2.31) give $u(0,0) = g_0(0)/\alpha_0 = h_0(0)/\gamma_0$. This implies the consistency condition $\gamma_0 g_0(0) = \alpha_0 h_0(0)$. Similarly there are three other consistency conditions at the corners $(1,0)$, $(0,1)$, $(1,1)$ given, respectively, by $\alpha_1 h_0(1) = \gamma_0 g_1(0)$, $\gamma_1 g_0(1) = \alpha_0 h_1(0)$, and $\gamma_1 g_1(1) = \alpha_1 h_1(1)$. In the case of general mixed linear boundary conditions of the form (2.29)-(2.32) similar consistency conditions arise. For example, it is easy to show that

$$V_{\alpha_0,\beta_0} H_{\gamma_0,\delta_0} u(x_1, x_2) = H_{\gamma_0,\delta_0} V_{\alpha_0,\beta_0} u(x_1, x_2)$$

for all x_1, x_2. If we put $x_1 = x_2 = 0$, then (2.29), (2.31) give

$$\alpha_0 h_0(0) - \beta_0 h_0'(0) = \gamma_0 g_0(0) - \delta_0 g_0'(0). \tag{2.33}$$

In the same way we can derive three other consistency conditions at $(1,0)$, $(0,1)$, $(1,1)$ respectively:

$$\gamma_0 g_1(0) - \delta_0 g_1'(0) = \alpha_1 h_0(1) - \beta_1 h_0'(1), \tag{2.34}$$

$$\gamma_1 g_0(1) - \delta_1 g_0'(1) = \alpha_0 h_1(0) - \beta_0 h_1'(0), \tag{2.35}$$

$$\gamma_1 g_1(1) - \delta_1 g_1'(1) = \alpha_1 h_1(1) - \beta_1 h_1'(1). \tag{2.36}$$

Let $\hat{\mathcal{M}}_i(r_i,\pi_i)$ be the space of piece polynomial functions defined by

$$\hat{\mathcal{M}}_i(r_i,\pi_i) = \{v \in C^1[0,1] : v|_{[x_i^{(j-1)},x_i^{(j)}]} \in P_{r_i},\ j = 1,\dots N_i\},\ i = 1,2. \tag{2.37}$$

Note that the restrictions $v(0) = v(1) = 0$ have been removed. The dimension of $\hat{\mathcal{M}}_i(r_i, \pi_i)$ is $M_i + 2$, where $M_i = N_i(r_i - 1)$, $i = 1, 2$. Let $\{\phi_{n_i}^{(i)}(x)\}_{n_i=0}^{M_i+1}$ be the B-spline type basis for $\hat{\mathcal{M}}_i(r_i, \pi_i)$. Assume that for $i = 1$ the first two basis functions satisfy

$$\alpha_0\phi_0^{(1)}(0) - \beta_0[\phi_0^{(1)}]'(0) = 1, \alpha_0\phi_1^{(1)}(0) - \beta_0[\phi_1^{(1)}]'(0) = 0, \tag{2.38}$$

and the last two basis functions are assumed to satisfy

$$\alpha_1\phi_{M_1}^{(1)}(1) - \beta_1[\phi_{M_1}^{(1)}]'(1) = 0, \alpha_1\phi_{M_1+1}^{(1)}(1) - \beta_1[\phi_{M_1+1}^{(1)}]'(1) = 1. \tag{2.39}$$

For $i = 2$ the first two basis functions satisfy

$$\gamma_0\phi_0^{(2)}(0) - \delta_0[\phi_0^{(2)}]'(0) = 1, \ \gamma_0\phi_1^{(2)}(0) - \delta_0[\phi_1^{(2)}]'(0) = 0, \tag{2.40}$$

and the last two basis functions satisfy

$$\gamma_1\phi_{M_2}^{(2)}(1) - \delta_1[\phi_{M_2}^{(2)}]'(1) = 0, \ \gamma_1\phi_{M_2+1}^{(2)}(1) - \delta_1[\phi_{M_2+1}^{(2)}]'(1) = 1. \tag{2.41}$$

We also assume that

$$\phi_{n_i}^{(i)}(0) = [\phi_{n_i}^{(i)}]'(0) = 0, \text{ and } \phi_{n_i}^{(i)}(1) = [\phi_{n_i}^{(i)}]'(1) = 0, \tag{2.42}$$

where $n_i = 2, \cdots, M_i - 1$, $i = 1, 2$. If we write the orthogonal spline collocation solution U $(\in \hat{\mathcal{M}}_1(r_1, \pi_1) \otimes \hat{\mathcal{M}}_2(r_2, \pi_2))$ as

$$U(x_1, x_2) = \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2}\phi_{n_1}^{(1)}(x_1)\phi_{n_2}^{(2)}(x_2), \tag{2.43}$$

then we have a total of $(M_1+2)(M_2+2) = M_1M_2+2M_1+2M_2+4$ unknowns denoted by u_{n_1,n_2}, $n_1 = 0, \ldots, M_1+1$ and $n_2 = 0, \ldots, M_2+1$.

Collocation at M_1M_2 internal Gauss points of Ω gives the following M_1M_2 equations

$$(L_1 + L_2)U(\xi_1^{(m_1)}, \xi_2^{(m_2)}) = f(\xi_1^{(m_1)}, \xi_2^{(m_2)}), \ m_i = 1, \ldots, M_i, \ i = 1, 2. \tag{2.44}$$

Collocation at the Gauss points on the boundary gives the following $2M_1 + 2M_2$ equations

$$\alpha_0 U(0,\xi_2^{(m_2)}) - \beta_0 U_{x_1}(0,\xi_2^{(m_2)}) = g_0(\xi_2^{(m_2)}), \tag{2.45}$$

$$\alpha_1 U(1,\xi_2^{(m_2)}) - \beta_1 U_{x_1}(1,\xi_2^{(m_2)}) = g_1(\xi_2^{(m_2)}), \tag{2.46}$$

$$\gamma_0 U(\xi_1^{(m_1)},0) - \delta_0 U_{x_2}(\xi_1^{(m_1)},0) = h_0(\xi_1^{(m_1)}), \tag{2.47}$$

$$\gamma_1 U(\xi_1^{(m_1)},1) - \delta_1 U_{x_2}(\xi_1^{(m_1)},1) = h_1(\xi_1^{(m_1)}). \tag{2.48}$$

$m_i = 1,\ldots,M_i$, $i = 1,2$. Now we have $M_1M_2 + 2M_1 + 2M_2$ equations for $M_1M_2 + 2M_1 + 2M_2 + 4$ unknowns. To obtain the last four equations we could collocate at the four corners. However there is no unique way of doing so. To collocate at $(0,0)$, for example, we could set

$$\alpha_0 U(0,0) - \beta_0 U_{x_1}(0,0) = g_0(0) \tag{2.49}$$

or

$$\gamma_0 U(0,0) - \delta_0 U_{x_2}(0,0) = h_0(0). \tag{2.50}$$

In a similar manner, we appear also to have a choice from two collocation equations at each of the corners $(1,0)$, $(0,1)$, $(1,1)$. To resolve this ambiguity, we will, instead impose on U the same consistency conditions at the corners as we have on u. For example, at $(0,0)$, we require

$$\begin{aligned} H_{\gamma_0,\delta_0} V_{\alpha_0,\beta_0} U(0,0) &= \alpha_0 h_0(0) - \beta_0 h_0'(0) \\ &= V_{\alpha_0,\beta_0} H_{\gamma_0,\delta_0} U(0,0) \\ &= \gamma_0 g_0(0) - \delta_0 g_0'(0). \end{aligned} \tag{2.51}$$

From (2.43) we have

$$\begin{aligned} & H_{\gamma_0,\delta_0} V_{\alpha_0,\beta_0} U(x_1, x_2) \\ &= \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2} H_{\gamma_0,\delta_0} V_{\alpha_0,\beta_0} \phi_{n_1}^{(1)}(x_1) \phi_{n_2}^{(2)}(x_2) \\ &= \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2} H_{\gamma_0,\delta_0} [\alpha_0 \phi_{n_1}^{(1)}(x_1) - \beta_0 [\phi_{n_1}^{(1)}]'(x_1)] \phi_{n_2}^{(2)}(x_2) \end{aligned}$$

$$= \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2}[\alpha_0\phi_{n_1}^{(1)}(x_1) - \beta_0[\phi_{n_1}^{(1)}]'(x_1)][\gamma_0\phi_{n_2}^{(2)}(x_2) - \delta_0[\phi_{n_2}^{(2)}]'(x_2)] \quad (2.52)$$

Putting $x_1 = x_2 = 0$ and using the fact that

$$\phi_j^{(i)}(0) = [\phi_j^{(i)}]'(0) = 0, \quad j = 2, \cdots, M_i + 1, \quad i = 1, 2,$$

we can reduced (2.52) to

$$\sum_{n_1=0}^{1} \sum_{n_2=0}^{1} u_{n_1,n_2}[\alpha_0\phi_{n_1}^{(1)}(x_1) - \beta_0[\phi_{n_1}^{(1)}]'(x_1)][\gamma_0\phi_{n_2}^{(2)}(x_2) - \delta_0[\phi_{n_2}^{(2)}]'(x_2)]. \quad (2.53)$$

Finally, applying (2.38), (2.40) we see that (2.53) reduced to $u_{0,0}$ alone. Thus

$$u_{0,0} = \alpha_0 h_0(0) - \beta_0 h_0'(0). \quad (2.54)$$

Similar arguments at the other corners give

$$u_{0,M_2+1} = \gamma_1 g_0(1) - \delta_1 g_0'(1), \quad (2.55)$$

$$u_{M_1+1,0} = \gamma_0 g_1(0) - \delta_0 g_1'(0), \quad (2.56)$$

$$u_{M_1+1,M_2+1} = \alpha_1 h_1(1) - \beta_1 h_1'(1). \quad (2.57)$$

We let U in the form of (2.43) satisfy the boundary conditions (2.29)-(2.32) at the Gaussian points and let $u_{0,0}$, u_{0,M_2+1}, $u_{M_1+1,0}$, u_{M_1+1,M_2+1} be given by (2.54)-(2.57). Then from (2.45) we have

$$\alpha_0 U(0, \xi_2^{(m_2)}) - \beta_0 U_{x_1}(0, \xi_2^{(m_2)}) = g_0(\xi_2^{(m_2)}). \quad (2.58)$$

Here the interpolation at Gaussian points takes place. Equation (2.58) is equivalent to

$$\alpha_0 \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2}\phi_{n_1}^{(1)}(0)\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) - \beta_0 \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2}[\phi_{n_1}^{(1)}]'(0)\phi_{n_2}^{(2)}(\xi_2^{(m_2)})$$
$$= g_0(\xi_2^{(m_2)}). \quad (2.59)$$

Since $\phi_{n_1}^{(1)}(0) = [\phi_{n_1}^{(1)}]'(0) = 0$, $n_1 = 2, \cdots, M_1 + 1$, and the second equation in (2.38), equation (2.59) can be simplified to

$$\sum_{n_2=0}^{M_2+1} u_{0,n_2}\phi_{n_2}^{(2)}(\xi_2^{(m_2)})(\alpha_0\phi_0^{(1)}(0) - \beta_0[\phi_0^{(1)}]'(0)) = g_0(\xi_2^{(m_2)}). \quad (2.60)$$

Using the first equation in (2.38) and extracting the first and last terms from the summation and moving them to the right hand side, we can then write equation (2.60) as

$$\sum_{n_2=1}^{M_2} u_{0,n_2}\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) = g_0(\xi_2^{(m_2)}) - u_{0,0}\phi_0^{(2)}(\xi_2^{(m_2)}) - u_{0,M_2+1}\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}). \tag{2.61}$$

Similarly we have,

$$\sum_{n_2=1}^{M_2} u_{M_1+1,n_2}\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) = g_1(\xi_2^{(m_2)}) - u_{M_1+1,0}\phi_0^{(2)}(\xi_2^{(m_2)}) - u_{M_1+1,M_2+1}\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}), \tag{2.62}$$

$$\sum_{n_1=1}^{M_1} u_{n_1,0}\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) = h_0(\xi_1^{(m_1)}) - u_{0,0}\phi_0^{(1)}(\xi_1^{(m_1)}) - u_{M_1+1,0}\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}), \tag{2.63}$$

$$\sum_{n_1=1}^{M_1} u_{n_1,M_2+1}\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) = h_1(\xi_1^{(m_1)}) - u_{0,M_2+1}\phi_0^{(1)}(\xi_1^{(m_1)}) - u_{M_1+1,M_2+1}\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}). \tag{2.64}$$

The coefficients u_{0,n_2}, u_{M_1+1,n_2}, $u_{n_1,0}$ and u_{n_1,M_2+1}, $n_i = 1,\ldots,M_i$, $i = 1,2$, can be obtained by solving equations (2.61)-(2.64).

To solve for the remaining unknowns, let $\vec{u}$ and $\vec{f}$ as in (2.8) and (2.9), respectively, but with

$$\begin{aligned} f_{m_1,m_2} &= f(\xi_1^{(m_1)},\xi_2^{(m_2)}) - L_1\phi_0^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{0,n_2}\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\ &- \phi_0^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{0,n_2}L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\ &- L_1\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{M_1+1,n_2}\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\ &- \phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{M_1+1,n_2}L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \end{aligned}$$

$$
\begin{aligned}
&- L_2\phi_0^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,0}\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- \phi_0^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,0}L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- L_2\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1}\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- \phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1}L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}). \qquad (2.65)
\end{aligned}
$$

From (2.61)-(2.64), (2.65) can be simplified as

$$
\begin{aligned}
f_{m_1,m_2} &= f(\xi_1^{(m_1)},\xi_2^{(m_2)}) - L_1\phi_0^{(1)}(\xi_1^{(m_1)})g_0(\xi_2^{(m_2)}) \\
&- \phi_0^{(1)}(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{0,n_2}L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- L_1\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})g_1(\xi_2^{(m_2)}) \\
&- \phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{M_1+1,n_2}L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- L_2\phi_0^{(2)}(\xi_2^{(m_2)})h_0(\xi_1^{(m_1)}) \\
&- \phi_0^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,0}L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- L_2\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)})h_1(\xi_1^{(m_1)}) \\
&- \phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1}L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}), \qquad (2.66)
\end{aligned}
$$

$m_i = 1,\ldots,M_i$, $i = 1,2$. The linear system for the remaining unknowns, $\vec{u}$, can be written as (2.10), where A_i, B_i, $i = 1,2$, are defined as (2.11)-(2.12) and with $\vec{f}$ modified as in (2.66).

The following lemma ensures that we can still use the Algorithm 2.1 in Section 3 to solve the linear system in the form (2.10) with $\vec{f}$ modified as in (2.66).

Lemma 2.3 *Let $\hat{\mathcal{M}}_1^0(r_1,\pi_1)$ be defined as*

$$
\hat{\mathcal{M}}_i^0(r_1,\pi_1) = \{v \in \hat{\mathcal{M}}_1(r_1,\pi_1) : \alpha_0 v(0) - \beta_0 v'(0) = 0,\ \alpha_1 v(1) - \beta_1 v'(1) = 0\}, \quad (2.67)
$$

where $\hat{\mathcal{M}}_1(r_1,\pi_1)$ is defined by (2.37). Let $\{\phi_{n_1}^{(1)}\}_{n_1=1}^{M_1}$ be a basis for $\hat{\mathcal{M}}_1^0(r_1,\pi_1)$ and let A_1 and B_1 be defined as (2.11) and (2.12), respectively, and W and D be defined as in Lemma 2.1. Let $F_1 = B_1^T W D B_1$, $G_1 = B_1^T W D A_1$. Then F_1 is a symmetric positive definite matrix and G_1 is a symmetric matrix.

To prove Lemma 2.3, we need the follow lemma which is Lemma 3.1 in [24] and Lemma 2.4.

Lemma 2.4 *For all $f, g \in \hat{\mathcal{M}}_1(r_1,\pi_1)$, let $< \cdot, \cdot >$ denote the quadratic form defined as*

$$< f, g >= \sum_{j=1}^{N_1} h_j \sum_{k=1}^{r_1-1} w_k f(\xi_1^{((j-1)(r_1-1)+k)}) g(\xi_1^{((j-1)(r_1-1)+k)}).$$

Then

$$< f, g'' >= -(f', g') + f(1)g'(1) - f(0)g'(0) - P_{r_1} \sum_{j=1}^{N_1} f_j^{(r_1)} g_j^{(r_1)} (h_1^{(j)})^{2r_1-1}, \quad (2.68)$$

where $f_j^{(r_1)}$, $g_j^{(r_1)}$ are the (r_1)th derivatives of f and g on $[x_1^{(j-1)}, x_1^{(j)}]$ respectively, P_{r_1} is a positive constant depending on r_1 only, $(\cdot,\cdot)$ is the standard inner product and $h_1^{(j)} = x_1^{(j)} - x_1^{(j-1)}$, $i = 1, \cdots, N_1$.

Next we give the proof of Lemma 2.3. The proof is essentially the same as Lemma 5.1 in [11], the only difference being the consideration of more general boundary conditions.

Proof: Clearly F_1 is symmetric. Since the diagonal elements of the matrices WD are positive and B_1 is nonsingular, then F_1 is positive definite.

To show G_1 is symmetric, first note that it is easily shown that,

$$G_1 = B_1^T W \tilde{A}_1 + B_1^T W \tilde{D} B_1,$$

where $\tilde{A}_1 = (\tilde{a}_{mn}^{(1)})_{m,n=1}^{M_1}$, $\tilde{a}_{mn}^{(1)} = -[\phi_n^{(1)}]''(\xi_1^{(m)})$ and $\tilde{D}$ is a diagonal matrix with diagonal element $\tilde{d}_{i,i} = c_1(\xi_1^{(i)})/a_1(\xi_1^{(i)})$, $i = 1, \cdots, M_1$.

Clearly we just need to show that $B_1^T W \tilde{A}_1$ is symmetric, which is equivalent to showing that

$$< \phi_i^{(1)}, [\phi_j^{(1)}]'' >=< [\phi_j^{(1)}]'', \phi_i^{(1)} >, \tag{2.69}$$

for $i = 1, \cdots, M_1$, $j = 1, \cdots, M_1$. By Lemma 2.4, equation (2.69) is equivalent to

$$\phi_i^{(1)}(1)[\phi_j^{(1)}]'(1) - \phi_i^{(1)}(0)[\phi_j^{(1)}]'(0) = \phi_j^{(1)}(1)[\phi_i^{(1)}]'(1) - \phi_j^{(1)}(0)[\phi_i^{(1)}]'(0). \tag{2.70}$$

The above equation follows from Lemma 1.1. ■

Next we show that the treatment for solving (2.1) with homogeneous Dirichlet boundary conditions given in Section 2.2 is just a special case of our treatment for more general boundary conditions. Suppose we are solving (2.1) with homogeneous Dirichlet boundary conditions as in (2.2). Let

$$U(x_1, x_2) = \sum_{n_1=0}^{M_1+1} \sum_{n_2=0}^{M_2+1} u_{n_1,n_2} \phi_{n_1}^{(1)}(x_1) \phi_{n_2}^{(2)}(x_2)$$

be the orthogonal spline collocation solution, where $\{\phi_{n_i}^{(i)}\}_{n_i=0}^{M_i+1}$, $i = 1, 2$ are basis functions for $\hat{\mathcal{M}}_i(r_i, \pi_i)$. From equations (2.2) and (2.54)-(2.57), we have $u_{0,0} = u_{0,M_2+1} = u_{M_1+1,0} = u_{M_1+1,M_2+1} = 0$. Consequently equation (2.61) becomes

$$\sum_{n_2=1}^{M_2} \phi_{n_2}^{(2)}(\xi_2^{(m_2)}) u_{0,n_2} = 0. \tag{2.71}$$

$m_2 = 1, \ldots, M_2$. Since the coefficient matrix of the linear system (2.71) is nonsingular, then

$$u_{0,n_2} = 0, \; n_2 = 1, \ldots, M_2.$$

Similarly we have

$$u_{M_1+1,n_2} = 0, \; n_2 = 1, \ldots, M_2,$$
$$u_{n_1,0} = u_{n_1,M_2+1} = 0, \; n_1 = 1, \ldots, M_1.$$

Hence

$$U(x_1, x_2) = \sum_{n_1=1}^{M_1} \sum_{n_2=1}^{M_2} u_{n_1,n_2} \phi_{n_1}^{(1)}(x_1) \phi_{n_2}^{(2)}(x_2).$$

Note that $\{\phi_{n_i}^{(i)}\}_{n_i=1}^{M_i}$, $i = 1,2$ are basis functions for $\mathcal{M}_i(r_i, \pi_i)$ defined as (2.5), and therefore this U is exactly the same as the solution obtained by the procedures given in Section 2 for solving (2.1).

Next we give a procedure for determining the B-spline type basis functions for $\hat{\mathcal{M}}_i(r_i, \pi_i)$, $i = 1,2$, so that (2.38)-(2.42) are satisfied. The basis functions $\{\phi_{n_i}^{(i)}\}_{n_i=2}^{M_i-1}$, $i = 1,2$, are the standard B-spline basis functions for $\hat{\mathcal{M}}_1(r_i, \pi_i)$, $i = 1,2$, so (2.42) is satisfied. The only problem here is to determine $\phi_0^{(i)}$, $\phi_1^{(i)}$, $\phi_{M_i}^{(i)}$, $\phi_{M_i+1}^{(i)}$. Since the procedures for $\hat{\mathcal{M}}_1(r_1, \pi_1)$ and $\hat{\mathcal{M}}_2(r_2, \pi_2)$ are the same, we give only that for $\hat{\mathcal{M}}_1(r_1, \pi_1)$ and state the result for $\hat{\mathcal{M}}_2(r_2, \pi_2)$.

Let $B_0^{(1)}, \ldots, B_{M_1+1}^{(1)}$ denote the B-spline basis functions for $\hat{\mathcal{M}}_1(r_1, \pi_1)$, where $B_0^{(1)}$, $\ldots$, $B_{r_1}^{(1)}$ are the B-splines on the first subinterval of π_1 and $B_{M_1+1-r_1}^{(1)}, \ldots, B_{M_1+1}^{(1)}$ are the B-splines on the last subinterval of π_1. The first two and last two basis functions, $\phi_0^{(1)}$, $\phi_1^{(1)}$, $\phi_{M_1}^{(1)}$, $\phi_{M_1+1}^{(1)}$, can be expressed in terms of the first two and last two B-spline functions. Following the approach in [11], we define

$$\phi_0^{(1)} = \phi_{0,0}^{(1)} B_0^{(1)} + \phi_{0,1}^{(1)} B_1^{(1)}, \tag{2.72}$$

$$\phi_1^{(1)} = \phi_{1,0}^{(1)} B_0^{(1)} + \phi_{1,1}^{(1)} B_1^{(1)}, \tag{2.73}$$

$$\phi_{M_1}^{(1)} = \phi_{M_1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1,1}^{(1)} B_{M_1+1}^{(1)}, \tag{2.74}$$

$$\phi_{M_1+1}^{(1)} = \phi_{M_1+1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1+1,1}^{(1)} B_{M_1+1}^{(1)}, \tag{2.75}$$

where $\phi_{0,0}^{(1)}$, $\phi_{0,1}^{(1)}$, $\phi_{1,0}^{(1)}$, $\phi_{1,1}^{(1)}$, $\phi_{M_1,0}^{(1)}$, $\phi_{M_1,1}^{(1)}$, $\phi_{M_1+1,0}^{(1)}$, $\phi_{M_1+1,1}^{(1)}$, are scalar constants. Putting expressions (2.72)-(2.75) into (2.38)-(2.41), we have

$$\alpha_0(\phi_{0,0}^{(1)} B_0^{(1)} + \phi_{0,1}^{(1)} B_1^{(1)})(0) - \beta_0(\phi_{0,0}^{(1)} B_0^{(1)} + \phi_{0,1}^{(1)} B_1^{(1)})'(0) = 1, \tag{2.76}$$

$$\alpha_0(\phi_{1,0}^{(1)} B_0^{(1)} + \phi_{1,1}^{(1)} B_1^{(1)})(0) - \beta_0(\phi_{1,0}^{(1)} B_0^{(1)} + \phi_{1,1}^{(1)} B_1^{(1)})'(0) = 0, \tag{2.77}$$

and

$$\alpha_1(\phi_{M_1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1,1}^{(1)} B_{M_1+1}^{(1)})(1) - \beta_1(\phi_{M_1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1,1}^{(1)} B_{M_1+1}^{(1)})'(1) = 0, \tag{2.78}$$

$$\alpha_1(\phi_{M_1+1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1+1,1}^{(1)} B_{M_1+1}^{(1)})(1) - \beta_1(\phi_{M_1+1,0}^{(1)} B_{M_1}^{(1)} + \phi_{M_1+1,1}^{(1)} B_{M_1+1}^{(1)})'(1) = 1. \tag{2.79}$$

By Lemma 1.1, we know that

$$B_0^{(1)}(0) \neq 0,\ [B_0^{(1)}]'(0) \neq 0,\ B_1^{(1)}(0) = 0,\ [B_1^{(1)}]'(0) \neq 0,$$
$$B_j^{(1)}(0) = [B_j^{(1)}]'(0) = 0,\ j = 2, \ldots, r_1,$$

and

$$B_j^{(1)}(1) = [B_j^{(1)}]'(1) = 0,\ j = M_1 + 1 - r_1, \ldots, M_1 - 1,$$
$$B_{M_1}^{(1)}(1) = 0,\ [B_{M_1}^{(1)}]'(1) \neq 0,\ B_{M_1+1}^{(1)}(1) \neq 0,\ [B_{M_1+1}^{(1)}]'(1) \neq 0.$$

Therefore equations (2.76) and (2.77) become

$$(\alpha_0 B_0^{(1)}(0) - \beta_0 [B_0^{(1)}]'(0))\phi_{0,0}^{(1)} - \beta_0 [B_1^{(1)}]'(0)\phi_{0,1}^{(1)} = 1 \tag{2.80}$$

and

$$(\alpha_0 B_0^{(1)}(0) - \beta_0 [B_0^{(1)}]'(0))\phi_{1,0}^{(1)} - \beta_0 [B_1^{(1)}]'(0)\phi_{1,1}^{(1)} = 0. \tag{2.81}$$

Let

$$\phi_{0,0}^{(1)} = \frac{\hat{\alpha}_0}{\hat{\alpha}_0^2 + \hat{\beta}_0^2}, \qquad \phi_{0,1}^{(1)} = -\frac{\hat{\beta}_0}{\hat{\alpha}_0^2 + \hat{\beta}_0^2}, \tag{2.82}$$

$$\phi_{1,0}^{(1)} = \frac{\hat{\beta}_0}{\hat{\alpha}_0^2 + \hat{\beta}_0^2}, \qquad \phi_{1,1}^{(1)} = \frac{\hat{\alpha}_0}{\hat{\alpha}_0^2 + \hat{\beta}_0^2}, \tag{2.83}$$

and

$$\hat{\alpha}_0 = \alpha_0 B_0^{(1)}(0) - \beta_0 [B_0^{(1)}]'(0),\ \hat{\beta}_0 = \beta_0 [B_1^{(1)}]'(0).$$

Then (2.80) and (2.81) are satisfied. Since $\alpha_0^2 + \beta_0^2 \neq 0$, then $\hat{\alpha}_0^2 + \hat{\beta}_0^2 \neq 0$, $\phi_{0,0}^{(1)}$, $\phi_{0,1}^{(1)}$, $\phi_{1,0}^{(1)}$ and $\phi_{1,1}^{(1)}$ are well defined. Similarly we choose $\phi_{M_1}^{(1)}$ and $\phi_{M_1+1}^{(1)}$ as defined by (2.74)-(2.75) with

$$\phi_{M_1,0}^{(1)} = \frac{\hat{\alpha}_1}{\hat{\alpha}_1^2 + \hat{\beta}_1^{\,2}}, \qquad \phi_{M_1,1}^{(1)} = \frac{\hat{\beta}_1}{\hat{\alpha}_1^2 + \hat{\beta}_1^{\,2}}, \tag{2.84}$$

$$\phi_{M_1+1,0}^{(1)} = -\frac{\hat{\beta}_1}{\hat{\alpha}_1^2 + \hat{\beta}_1^{\,2}}, \qquad \phi_{M_1+1,1}^{(1)} = \frac{\hat{\alpha}_1}{\hat{\alpha}_1^2 + \hat{\beta}_1^{\,2}} \tag{2.85}$$

and

$$\hat{\alpha}_1 = \alpha_1 B_{M_1+1,1}^{(1)}(1) - \beta_1 [B_{M_1+1,1}^{(1)}]'(1),\ \hat{\beta}_1 = \beta_1 [B_{M_1,1}^{(1)}]'(1),$$

to satisfy (2.39).

In the same way we can determine the basis functions for $\hat{\mathcal{M}}_2(r_2, \pi_2)$. The first two basis functions can be chosen as

$$\phi_0^{(2)} = \phi_{0,0}^{(2)} B_0^{(2)} + \phi_{0,1}^{(2)} B_1^{(2)},$$
$$\phi_1^{(2)} = \phi_{1,0}^{(2)} B_0^{(2)} + \phi_{1,1}^{(2)} B_1^{(2)},$$

where

$$\phi_{0,0}^{(2)} = \frac{\hat{\gamma}_0}{\hat{\gamma}_0^2 + \hat{\delta}_0^2}, \quad \phi_{0,1}^{(2)} = -\frac{\hat{\delta}_0}{\hat{\gamma}_0^2 + \hat{\delta}_0^2}, \tag{2.86}$$

$$\phi_{1,0}^{(2)} = \frac{\hat{\delta}_0}{\hat{\gamma}_0^2 + \hat{\delta}_0^2}, \quad \phi_{1,1}^{(2)} = \frac{\hat{\gamma}_0}{\hat{\gamma}_0^2 + \hat{\delta}_0^2}, \tag{2.87}$$

and

$$\hat{\gamma}_0 = \gamma_0 B_0^{(2)}(0) - \delta_0 [B_0^{(2)}]'(0), \ \hat{\delta}_0 = \delta_0 [B_1^{(2)}]'(0).$$

The last two basis functions can be chosen as

$$\phi_{M_2}^{(2)} = \phi_{M_2,0}^{(2)} B_{M_2}^{(2)} + \phi_{M_2,1}^{(2)} B_{M_2+1}^{(2)},$$
$$\phi_{M_2+1}^{(2)} = \phi_{M_2+1,0}^{(2)} B_{M_2}^{(2)} + \phi_{M_2+1,1}^{(2)} B_{M_2+1}^{(2)},$$

where

$$\phi_{M_2,0}^{(2)} = \frac{\hat{\gamma}_1}{\hat{\gamma}_1^2 + \hat{\delta}_1^2}, \qquad \phi_{M_2,1}^{(2)} = \frac{\hat{\delta}_1}{\hat{\gamma}_1^2 + \hat{\delta}_1^2}, \tag{2.88}$$

$$\phi_{M_2+1,0}^{(2)} = -\frac{\hat{\delta}_1}{\hat{\gamma}_1^2 + \hat{\delta}_1^2}, \quad \phi_{M_2+1,1}^{(2)} = \frac{\hat{\gamma}_1}{\hat{\gamma}_1^2 + \hat{\delta}_1^2} \tag{2.89}$$

and

$$\hat{\gamma}_1 = \gamma_1 B_{M_2+1}^{(2)}(1) - \delta_1 [B_{M_2+1}^{(2)}]'(1), \ \hat{\delta}_1 = \delta_1 [B_{M_2}^{(2)}]'(1).$$

We have implemented the algorithm for general boundary conditions in the form of (2.29)-(2.32). The generalization from the homogeneous Dirichlet boundary case to general boundary condition case is straightforward since the matrix structure of the linear system does not change. The changes are in the routines to set up the collocation matrix and the routine to set up the right hand side vector.

We now show the performance of our implementation by solving two test problems. The first example is Examples 4 in Kaufman-Warner [41]. It is a Poisson's equation in polar coordinates having the form,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}=16r^2, \tag{2.90}$$

on the quarter disk, $0<r<1$, $0<\theta<\pi/2$, with boundary conditions,

$$\begin{aligned} &u(0,\theta)=0,\ \ u(1,\theta)=1-\cos(4\theta)\ \text{ for } 0<\theta<\pi/2,\\ &u_r(r,0)=0,\ \ u_r(r,\pi/2)=0\ \text{ for } 0<r<1. \end{aligned}$$

The solution to the above problem is $u(r,\theta)=r^4(1-\cos(4\theta))$. In order to use our program, we make the following changes in variables: $x_1=\frac{2\theta}{\pi}$, $x_2=r$, $v(x_1,x_2)=u(r,\theta)$. Then equation becomes

$$-\left(\frac{2}{\pi}\right)^2\frac{\partial^2 v}{\partial x_1^2}-x_2^2\frac{\partial^2 v}{\partial x_2^2}-x_2\frac{\partial v}{\partial x_2}=-16x_2^4. \tag{2.91}$$

The domain is now $0\le x_1,x_2\le 1$ and the boundary conditions become

$$\left.\begin{aligned} &v(x_1,0)=0, \quad v(x_1,1)=1-\cos(2\pi x_1),\\ &v_{x_1}(0,x_2)=0, \quad v_{x_1}(1,x_2)=0. \end{aligned}\right\} \tag{2.92}$$

The solution is $x_2^4(1-\cos(2\pi x_1))$.

In Table 2.8 we present the results based on (2.91) for different values M and k, where M is the number of subintervals and k is the number of collocation points in each subintervals. We use $E_{i,u}$, $i=1,2$, to denote the L_∞ error for $\frac{\partial u}{\partial x_i}$, $i=1,2$, and $E_{i,m}$, $i=1,2$, to denote the maximum error for $\frac{\partial u}{\partial x_i}$, $i=1,2$, at mesh points. The L_∞ errors for v are computed on a 100×100 uniform grid and the computations are carried out on a Sparc 1000 workstation. Time is given in seconds. The L_∞ errors behave like $O(h^{k+2})$ for $k=2$, 3, and 4, respectively, while the L_∞ errors for u_{x_1} and u_{x_2} behave like $O(h^{k+1})$ for these k values. Therefore the convergence rate of the L_∞ error for the derivative is one less than the convergence rate for the function. We also observed this same phenomena for the problem with homogeneous Dirichlet

boundary conditions. The convergence rates of L_∞ errors shown here agree with the theoretically results, [4], [12]. On the other hand, it appears that the maximum errors at mesh points behave differently from that of the problems with homogeneous Dirichlet boundary conditions. For example, for $k = 3$, we observe that the errors behave only like $O(h^5)$ while for the homogeneous Dirichlet case the errors behave like $O(h^6)$.

k	M	L_∞ Error			Max. Error at Mesh Points		
		E_u	$E_{1,u}$	$E_{2,u}$	E_m	$E_{1,m}$	$E_{2,m}$
2	4	4.96-03	1.25-01	3.18-02	4.71-03	1.14-01	1.99-02
2	8	4.70-04	2.12-02	2.94-03	3.95-04	8.71-03	2.94-03
2	16	3.29-05	2.86-03	4.24-04	2.67-05	5.76-04	4.24-04
2	32	1.82-06	3.36-04	6.01-05	1.70-06	3.65-05	6.01-05
2	64	1.25-07	4.50-05	8.40-06	1.07-07	2.29-06	8.40-06
3	4	3.56-04	1.30-02	3.89-03	1.81-04	1.30-02	1.95-03
3	8	1.54-05	9.52-04	3.56-04	2.89-06	9.52-04	5.19-05
3	16	5.55-07	6.43-05	2.52-05	4.54-08	6.15-05	1.51-06
3	32	1.87-08	3.88-06	1.53-06	7.10-10	3.88-06	4.94-08
3	64	6.40-10	2.55-07	1.09-07	4.37-11	2.43-07	1.17-08
4	4	2.26-05	9.27-04	2.74-04	1.08-05	7.67-04	1.50-04
4	8	5.17-07	3.72-05	7.03-06	2.15-07	1.34-05	6.82-06
4	16	9.32-09	1.22-06	2.33-07	3.56-09	2.16-07	2.33-07
4	32	1.40-10	4.53-08	2.07-08	7.35-11	3.51-09	2.07-08
4	64	1.69-10	5.88-09	5.74-08	1.65-10	1.42-08	5.74-08

Table 2.8: The errors in solving Equation (2.91) with boundary conditions (2.92) using ELLDCM.

We conclude that the convergence of the maximum error at mesh points is dependent on the boundary conditions of the differential equation. The reason for the

drop in the rate of convergence with general boundary conditions is that we perform interpolation of function values at Gaussian points for the boundary conditions. In fact convergence depends on how non-homogeneous boundary conditions are approximated, see [11].

The second example is also a Poisson's equation with mixed boundary conditions. The reason for using this example is that the convergence rate is clearly demonstrated by the result. The problem is as follows:

$$-\frac{\partial^2 u}{\partial x_1^2} - \frac{\partial^2 u}{\partial x_2^2} = f \tag{2.93}$$

The boundary conditions are

$$\left.\begin{array}{ll} u(0,x_2) - u_{x_1}(0,x_2) = x_2^m - x_2^{m-1} + 1, & u(1,x_2) = 2 + x_2^{m-1} + x_2^m, \\ u(x_1,0) - u_{x_2}(x_1,0) = x_1^m + 1, & u(x_1,1) = x_1^m + x_1 + 2, \end{array}\right\} \tag{2.94}$$

The function f is chosen such that the solution $u = x_1^m + x_2^m + x_1 x_2^{m-1} + 1$, where m is an integer and $m \geq 3$.

First we solve the above problem for $m = 5$. In Table 2.9 we report the results. We solve the problem with $k = 2$ and $k = 3$. The errors are zero for any larger k value since the degree of the approximate solution is higher than the degree of the solution. The convergence rate of the maximum error at the mesh points is the same as the convergence rate of the L_∞ error.

We also observed that the convergence rate of the maximum error at the mesh points is the same as that of the L_∞ error for $m = 6$. The numerical results given above further support our claim that the convergence of the maximum error at mesh points is dependent on the boundary conditions presented.

k	M	L_∞ Error			Max. Error at Mesh Points		
		E_u	$E_{1,u}$	$E_{2,u}$	E_m	$E_{1,m}$	$E_{2,m}$
2	4	1.91-03	1.60-02	1.76-02	4.68-04	1.31-02	1.61-02
2	8	1.26-04	2.07-03	2.26-03	3.35-05	1.76-03	2.13-03
2	16	8.10-06	2.65-04	2.88-04	2.26-06	2.27-04	2.74-04
2	32	4.64-07	2.89-05	3.47-05	1.49-07	2.89-05	3.47-05
3	4	1.77-05	2.68-04	2.68-04	3.69-06	2.02-04	2.02-04
3	8	5.54-07	1.67-05	1.67-05	1.16-07	1.26-05	1.26-05
3	16	1.71-08	9.82-07	9.81-07	3.66-09	7.92-07	7.92-07
3	32	5.85-10	5.92-08	5.91-08	1.57-10	4.95-08	4.95-08
4	4	9.33-13	1.01-11	2.20-11	7.82-13	1.47-12	2.20-11
4	8	4.08-11	6.92-10	1.86-09	3.69-11	7.01-11	1.86-09
4	16	2.62-10	8.39-09	2.27-08	2.49-10	5.97-10	2.27-08
4	32	9.43-11	6.37-09	1.55-08	9.23-11	8.05-10	1.55-08

Table 2.9: The errors in solving Equation (2.93) with boundary conditions (2.94) using ELLDCM, $m = 5$.

2.6 Concluding Remarks

In this chapter, we have examined an algorithm for the numerical solution of a family of linear separable elliptic PDEs based on orthogonal spline collocation and matrix decomposition. The parallel performance of this algorithm was investigated through numerical experiments on a multiple processor system, the Alliant/FX2800. As mentioned previously, the orthogonal spline collocation and matrix decomposition algorithm, described in this chapter, will be used as a major component within the method-of-lines algorithm for numerical solution of parabolic partial differential equations, to be described in the next chapter.

Chapter 3

Parabolic Case

3.1 The Method of Lines

The method of lines approach for solving time dependent initial value partial differential equations has received considerable attention recently, see e.g. [51], [55], [69]. The idea of this method is to discretize all but one of the independent variables, giving a system of ordinary differential equations with one independent variable. In the case of parabolic PDEs, where one variable is time and the others are space variables, discretizing the time results in a sequence of elliptic boundary value problems, related by some difference equations. This is called the *transverse* method of lines. Discretizing in the spatial variables results in a system of initial value ODEs, and is known as *longitudinal* method of lines. The distinction between the *transverse* and *longitudinal* method of lines was first made in [47]. We consider only the *longitudinal* method of lines here.

The method of lines has been successful in solving one space variable parabolic PDEs, see e.g. [51]. We will apply this method for solving two space dimension linear parabolic PDEs. When applying this method, careful consideration of the following three components is important,

(i) the spatial discretization,

(ii) the time-stepping procedure in the solution of the system of ODEs,

(iii) the linear equation solver.

For (i) the usual following choices are

- Finite difference methods,
- Finite element methods, such as L^2 Galerkin, H^{-1} Galerkin, etc,
- Collocation methods, such as orthogonal spline collocation.

As described in Chapter 1, orthogonal splines have several advantages. Compared to other choices, they are conceptually simpler, easer to implement and inherently parallel. For (ii), we choose the efficient differential/algebraic solver DASSL, [59], to solve the system of ODEs. We will discuss this choice further in the next section. For (iii), we perform a decomposition of the linear system into independent ABD subsystems and then solve each subsystem with an ABD solver, ARCECO. This will also be discussed further in the next section.

3.2 The Numerical Solution of Linear Parabolic PDEs

We consider here the following linear parabolic equation with homogeneous Dirichlet boundary conditions,

$$\frac{\partial u}{\partial t} = (L_1 + L_2)u + f(x_1, x_2, t), \quad (x_1, x_2, t) \in \Omega \times [0, T], \tag{3.1}$$

$$u(x_1, x_2, t) = 0, \quad (x_1, x_2) \in \partial\Omega, \tag{3.2}$$

$$u(x_1, x_2, 0) = \theta(x_1, x_2), \quad (x_1, x_2) \in \Omega, \tag{3.3}$$

where

$$L_1 = a_1(x_1)\frac{\partial^2}{\partial x_1^2} + c_1(x_1), \quad a_1(x_1) > 0 \text{ in } [0, 1], \tag{3.4}$$

$$L_2 = a_2(x_2)\frac{\partial^2}{\partial x_2^2} + b_2(x_2)\frac{\partial}{\partial x_2} + c_2(x_2), \quad a_2(x_2) > 0 \text{ in } [0,1], \tag{3.5}$$

$\Omega = [0, 1] \times [0, 1]$.

We require the orthogonal spline collocation approximation $U \in \mathcal{M}_1(r_1, \pi_1) \otimes \mathcal{M}_2(r_2, \pi_2)$, with $\mathcal{M}_i(r_i, \pi_i)$, $i = 1, 2$, defined as in (2.5), to satisfy

$$\frac{\partial U}{\partial t}(\xi_1^{(m_1)}, \xi_2^{(m_2)}, t) = (L_1 + L_2)U(\xi_1^{(m_1)}, \xi_2^{(m_2)}, t) + f(\xi_1^{(m_1)}, \xi_2^{(m_2)}, t),$$
$$m_i = 1, \ldots, M_i, \ t \in [0, T], \tag{3.6}$$

where $M_i = N_i(r_i - 1), i = 1, 2$.

Let $\{\phi_{n_i}^{(i)}\}_{n_i=1}^{M_i}$ be a set of basis functions for $\mathcal{M}_i(r_i, \pi_i)$. Suppose that

$$U(x_1, x_2, t) = \sum_{n_1=1}^{M_1} \sum_{n_2=1}^{M_2} u_{n_1,n_2}(t)\phi_{n_1}^{(1)}(x_1)\phi_{n_2}^{(2)}(x_2), \tag{3.7}$$

and set

$$\vec{u}(t) = [u_{1,1}(t), u_{1,2}(t), \ldots, u_{1,M_2}(t), \ldots, u_{M_1,1}(t), \ldots, u_{M_1,M_2}(t)]^T, \tag{3.8}$$

and

$$\vec{f}(t) = [f_{1,1}(t), f_{1,2}(t), \ldots, f_{1,M_2}(t), \ldots, f_{M_1,1}(t), \ldots, f_{M_1,M_2}(t)]^T, \tag{3.9}$$

where $f_{m_1,m_2}(t) = f(\xi_1^{(m_1)}, \xi_2^{(m_2)}, t)$. Then (3.6) can be written as

$$(B_1 \otimes B_2)\frac{d\vec{u}}{dt} = (A_1 \otimes B_2 + B_1 \otimes A_2)\vec{u} + \vec{f}(t), \ t \in [0, T], \tag{3.10}$$

where

$$A_i = (a_{m\,n}^{(i)})_{m,n=1}^{M_i}, \quad a_{mn}^{(i)} = L_i\phi_n^{(i)}(\xi_i^{(m)}), \tag{3.11}$$

$$B_i = (b_{m\,n}^{(i)})_{m,n=1}^{M_i}, \quad b_{mn}^{(i)} = \phi_n^{(i)}(\xi_i^{(m)}), \ i = 1, 2. \tag{3.12}$$

It follows from Lemma 2.1 and Lemma 2.2 that $F_1 = B_1^T W D B_1$ is symmetric and positive definite and $G_1 = B_1^T W D A_1$ is symmetric, where W and D are defined in Lemma 2.1. Let I_1 and I_2 be identity matrices of order M_1 and M_2 respectively. Further, as before, there exists a nonsingular matrix Z such that

$$Z^T G_1 Z = \Lambda, \ Z^T F_1 Z = I_1, \tag{3.13}$$

where $\Lambda = diag(\lambda_i)_{i=1}^{M_1}$. Hence we have,

$$(Z^T B_1^T W D \otimes I_2)(A_1 \otimes B_2 + B_1 \otimes A_2)(Z \otimes I_2) = \Lambda \otimes B_2 + I_1 \otimes A_2.$$

As a result, from (3.10),

$$(B_1^T W D \otimes I_2)(B_1 \otimes B_2)\frac{d\vec{u}}{dt} = (B_1^T W D \otimes I_2)(A_1 \otimes B_2 + B_1 \otimes A_2)\vec{u} + (B_1^T W D \otimes I_2)\vec{f}(t),$$

which may be written,

$$(F_1 \otimes B_2)\frac{d\vec{u}}{dt} = (G_1 \otimes B_2 + F_1 \otimes A_2)\vec{u} + \vec{g}(t),$$

where $\vec{g}(t) = (B_1^T W D \otimes I_2)\vec{f}(t)$.

If we multiply both sides of the above equation by $(Z^T \otimes I_2)$, then,

$$(Z^T \otimes I_2)(F_1 \otimes B_2)(Z \otimes I_2)(Z^{-1} \otimes I_2)\frac{d\vec{u}}{dt} =$$
$$(Z^T \otimes I_2)(G_1 \otimes B_2 + F_1 \otimes A_2)(Z \otimes I_2)(Z^{-1} \otimes I_2)\vec{u} + (Z^T \otimes I_2)\vec{g}(t),$$

which by (3.13) is equivalent to

$$(I_1 \otimes B_2)\frac{d\vec{w}}{dt} = (\Lambda \otimes B_2 + I_1 \otimes A_2)\vec{w} + \vec{h}(t), \tag{3.14}$$

where $\vec{w}(t) = (Z^{-1} \otimes I_2)\vec{u}(t)$ and $\vec{h}(t) = ((Z^T B_1^T W D) \otimes I_2)\vec{f}(t)$.

Equation (3.14) gives a system of linear initial value ODEs. For some ODE solvers such as DSS/2, [62], [65], GEAR [31], or EPISODE, [32], [34], we would have to express $\frac{d\vec{w}}{dt}$ explicitly, by inverting B_2. There are some ODE solvers, e.g. GEARIB [33], LSODI [35], which can handle this type of equation directly. However in anticipation of a future extension of our algorithm involving the use of monomial basis functions for the space variables, we will use a differential/algebraic solver to solve equation (3.14). Differential/algebraic solvers such as DASSL [59] or the differential/algebraic solver in SPRINT [9] can handle equation (3.14) directly. Differential/algebraic equations of the form

$$A(t)\frac{d\vec{y}}{dt} = \vec{g}(t, \vec{y}),\ t \in [0, T], \tag{3.15}$$

where $A(t)$ is a matrix and $\vec{y}$ and $\vec{g}$ are both vector functions, are often encountered when applying the method of lines based on splines for solving parabolic differential equations. There are several issues that one should be aware of when trying to solve such systems. Many DAE systems can be solved using numerical methods for solving stiff ODE systems [28], but DAE systems may have some properties which are very different from those of ODE systems [60]. The initial conditions for the function and its derivative must be chosen to be consistent since error estimates used in the selection of stepsizes are sensitive to inconsistencies in the initial conditions. Also a DAE solver needs a more accurate approximation to the iteration matrix than a basic ODE solver.

One good choice for a DAE solver is DASSL. There are several reasons for this choice. First DASSL does not need to invert the left hand matrix and thus the ABD structure of the iteration matrix itself is preserved. In addition, DASSL is a well written and robust program; even in the current context where we are solving an ODE system, DASSL is just as efficient as any standard ODE solver. Furthermore, if we use a monomial basis [5] instead of the B-spline type basis then the resulting system of ODEs has the form

$$A(t)\frac{d\vec{y}}{dt} = B(t)\vec{y}(t) + \vec{f}(t), \tag{3.16}$$

where the matrix A(t) is not square. Together with this we have continuous conditions on $\vec{y}$. In such a case, only an DAE solver, such as DASSL, can handle the above system.

DASSL solves systems of differential/algebraic equations [15] of the form

$$\vec{F}(t, \vec{y}, \vec{y}') = 0, \quad \vec{y}(t_0) = \vec{y}_0, \quad \vec{y}'(t_0) = \vec{y}'_0, \tag{3.17}$$

where $\vec{F}$, $\vec{y}$ and $\vec{y}'$ are vectors of the same dimension. As described in [27], the basic idea for solving DAE systems is to replace the derivative $\vec{y}'$ in (3.17) by a difference approximation, and then to solve the resulting equation by Newton's method. If we replace the derivative in (3.17) by a simple first order backward difference formula

(BDF), then we have the following implicit Euler scheme

$$\vec{F}(t_{n+1}, \vec{y}_{n+1}, \frac{\vec{y}_{n+1} - \vec{y}_n}{\Delta t_{n+1}}) = 0, \tag{3.18}$$

where $\Delta t_{n+1} = t_{n+1} - t_n$. Then the resulting equation (3.18) is solved for $\vec{y}_{n+1}$ by using Newton's method. Let $\vec{y}_{n+1}^{(m)}$ be the mth Newton iterative approximation of y_{n+1}. Then Newton's iteration can be written as,

$$\vec{y}_{n+1}^{(m+1)} = \vec{y}_{n+1}^{(m)} - (G^{-1}F)(t_{n+1}, \vec{y}_{n+1}^{(m)}, \frac{\vec{y}_{n+1}^{(m)} - \vec{y}_n}{\Delta t_{n+1}}), \quad m = 0, 1, \cdots, \tag{3.19}$$

where

$$G = \left(\frac{\partial \vec{F}}{\partial \vec{y}} + \frac{1}{\Delta t_{n+1}} \frac{\partial \vec{F}}{\partial \vec{y}'} \right).$$

The algorithm implemented in DASSL uses a kth order backward difference formula, where k ranges from 1 to 5, to approximate the derivative. It chooses the order k and the step size Δt_{n+1} depending on the behavior of the solution over k previous steps ([15], [60], [61]). For our case, only one Newton's iteration will be performed since our problem is linear with respect to $\vec{y}_{n+1}$.

As we have seen, the spatial discretization is used to transform (3.1)-(3.3) into the ODE system (3.14). This system is solved using DASSL within the context of a generalization of Algorithm 2.1 as follows.

Given A_i, B_i, $i = 1, 2$, as (3.11), (3.12), W and D as in Lemma 2.1,
$F_1 = B_1^T W D B_1$ and $G_1 = B_1^T W D A_1$.

Step 1 Determine Λ and Z satisfying $Z^T G_1 Z = \Lambda$, $Z^T F_1 Z = I_1$.

Step 2 Compute $\vec{w}(0)$ and $\frac{d\vec{w}}{dt}(0)$.

Step 3 Use DASSL to solve
$(I_1 \otimes B_2)\frac{d\vec{w}}{dt} = (\Lambda \otimes B_2 + I_1 \otimes A_2)\vec{w} + \vec{h}(t)$
where $\vec{h}(t) = (Z^T B_1^T W D \otimes I_2)\vec{f}(t)$.

Step 4 Compute $\vec{u}(T) = (Z \otimes I_2)\vec{w}(T)$.

Algorithm 3.1

We need to pay particular attention to Step 3. The ODE system in Step 3 is decoupled, and we can solve it by solving in parallel each subsystem

$$B_2\frac{d\vec{w}_i}{dt} = (\lambda_i B_2 + A_2)\vec{w}_i + \vec{h}_i(t), \quad i = 1, \cdots, M_1, \; t \in [0, T],$$

where $\vec{w}(t) = [\vec{w}_1(t), \cdots, \vec{w}_{M_1}(t)]$ and $\vec{h}(t) = [\vec{h}_1(t), \cdots, \vec{h}_{M_1}(t)]$. But we should realize that in order to evaluate each $h_i(t)$, $i = 1, \cdots, M_1$, we need to compute the whole vector $\vec{f}(t)$. These computations can also be done in parallel, but we still need to do extra calculations. On the other hand, since some of the subsystems may need fewer time steps due to the magnitude of the eigenvalues and therefore fewer evaluations of $\vec{f}(t)$ for these subsystems, it may in fact be more efficient to do Step 3 in this way. Examining this decoupling approach in detail will be part of our future work.

For equation (3.14) in Step 3, the Jacobian matrix required by DASSL for each time step is,

$$(\Lambda - c * I_1) \otimes B_2 + I_1 \otimes A_2, \tag{3.20}$$

where c is a scalar determined by DASSL. In general the matrix in (3.20) decouples into matrices

$$A_2 + (\lambda_i - c)B_2, \quad i = 1, \cdots, M_1.$$

For example, for $k = 3$ and M = 4, the above matrix has the structure shown as Figure 3.1. Here we use k to denote the number of collocation points in each subinterval and M to denote the number of subintervals. The Jacobian matrix here can be decoupled into independent subsystems as indicated by the $\times$'s in Figure 3.1. Note that each subsystem has an ABD structure.

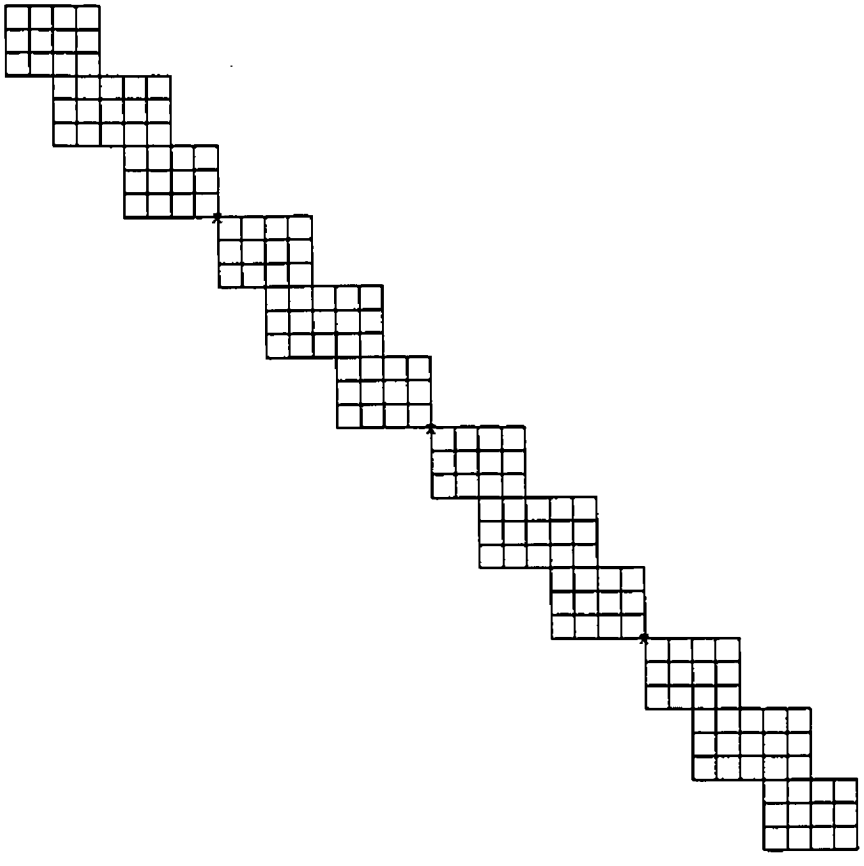

Figure 3.1: The structure of the Jacobian matrix.

DASSL has the option of treating the Jacobian matrix as a full or banded system. If the matrix of Fig. 3.1 is viewed as banded, then the band would contain the non-zero blocks as in Figure 3.2.

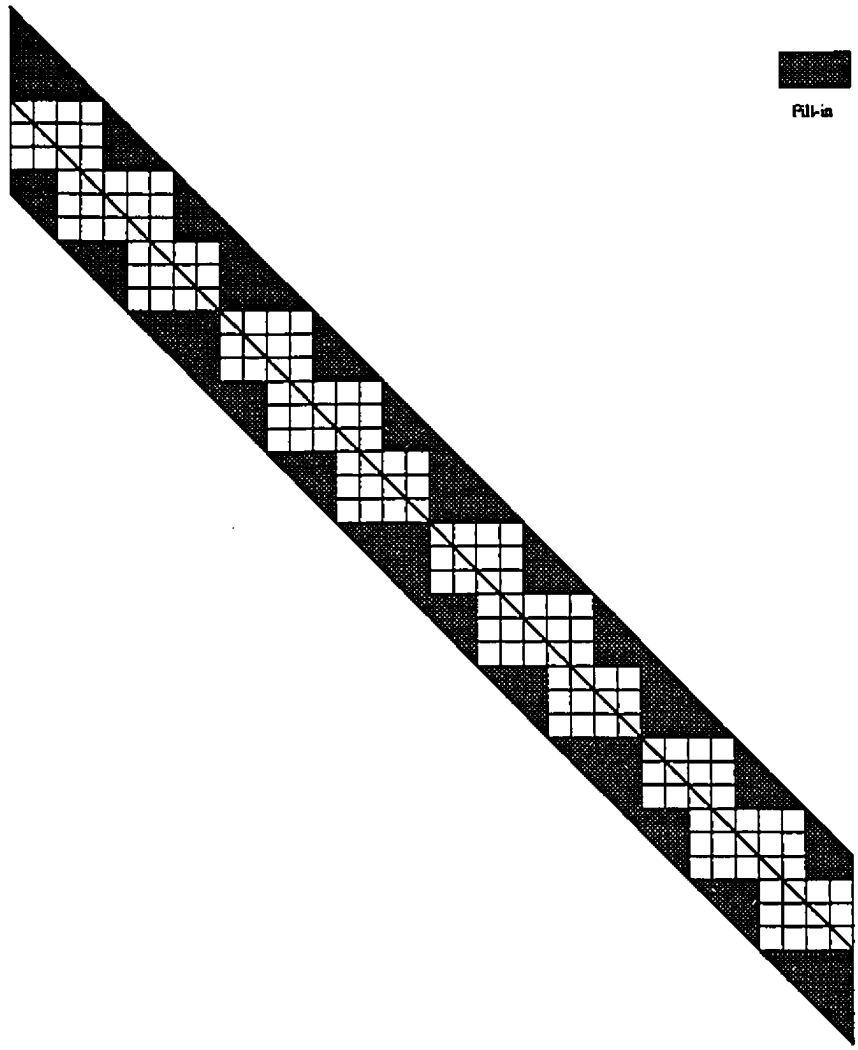

Figure 3.2: The structure of the banded Jacobian matrix.

Obviously there is substantial excess storage used for elements which are in fact known to be zero. This storage takes up to 50% percent of that of the nonzero elements. Since only the elements inside the small square boxes are nonzero, considerable fill-in occurs inside the band and even beyond the band when we include the fill-in introduced by Gaussian elimination, with partial pivoting, on a banded matrix.

It is better to consider modifying DASSL in order to solve (3.14) efficiently. The modifications are (i) decoupling the Jacobian matrix of the ODE system, (ii) augmenting the linear algebra features of DASSL to handle an ABD system structure within each decoupled subsystem. For each time step, we need to solve a linear system involving the Jacobian matrix and having the form,

$$((\Lambda - c * I_1) \otimes B_2 + I_1 \otimes A_2)\vec{x} = \vec{r}, \tag{3.21}$$

where $\vec{r}$ is a known vector and $\vec{x}$ is to be determined. The above linear system can be solved in parallel as in Chapter 2, that is the system can be divided into smaller independent systems which can be solved in parallel. Each smaller linear system has the form

$$((\lambda_i - c)B_2 + A_2)\vec{x_i} = \vec{r_i}, \tag{3.22}$$

in which the matrix has an almost block diagonal structure and thus can be factored by ARCECO.

In order to use DASSL to solve the ODE system arising in Step 3 of Algorithm 3.2, we need to supply the initial conditions for $\vec{w}$, i.e. $\vec{w}(0)$ and $\vec{w}'(0)$. Collocation of the initial condition (3.3) gives us

$$(B_1 \otimes B_2)\vec{u}(0) = \vec{\theta}, \tag{3.23}$$

where

$$\vec{\theta} = [\theta_{1,1}, \theta_{1,2}, \ldots, \theta_{1,M_2}, \ldots, \theta_{M_1,1}, \ldots, \theta_{M_1,M_2}]^T.$$

with $\theta_{m_1,m_2} = \theta(\xi_1^{(m_1)}, \xi_2^{(m_2)})$. Since $\vec{u}(0) = (Z^T \otimes I)\vec{w}(0)$, (3.23) yields

$$(B_1 Z \otimes B_2)\vec{w}(0) = \vec{\theta}.$$

From $Z^T B_1^T W D B_1 Z = I_1$, we have

$$(I_1 \otimes B_2)\vec{w}(0) = (Z^T B_1^T W D \otimes I_2)\vec{\theta}$$

which can be used to compute $\vec{w}(0)$. We give the following algorithm to compute $\vec{w}(0)$.

Step 1 Compute $\vec{\theta}^* = (Z^T B_1^T W D \otimes I_2)\vec{\theta}$.

Step 2 Solve $(I_1 \otimes B_2)\vec{w}(0) = \vec{\theta}^*$.

Algorithm 3.2

It is easy to see that both steps in Algorithm 3.2 are highly suitable for parallel computations. The computations for Step 1 are the same as those in Step 2 of Algorithm 3.1. Step 2 of Algorithm 3.2 can be performed by one call to ARCEDC, and M_1 calls to ARCESL.

Since, from (3.14),

$$(I_1 \otimes B_2)\frac{d\vec{w}(0)}{dt} = (\Lambda \otimes B_2 + I_1 \otimes A_2)\vec{w}(0) + \vec{h}(0), \tag{3.24}$$

after computing $\vec{w}(0)$, we can solve for $\frac{d\vec{w}(0)}{dt}$ from (3.24).

3.3 Modification of DASSL

We present a section of the calling tree for DASSL. This will be useful during the description of the modifications to DASSL.

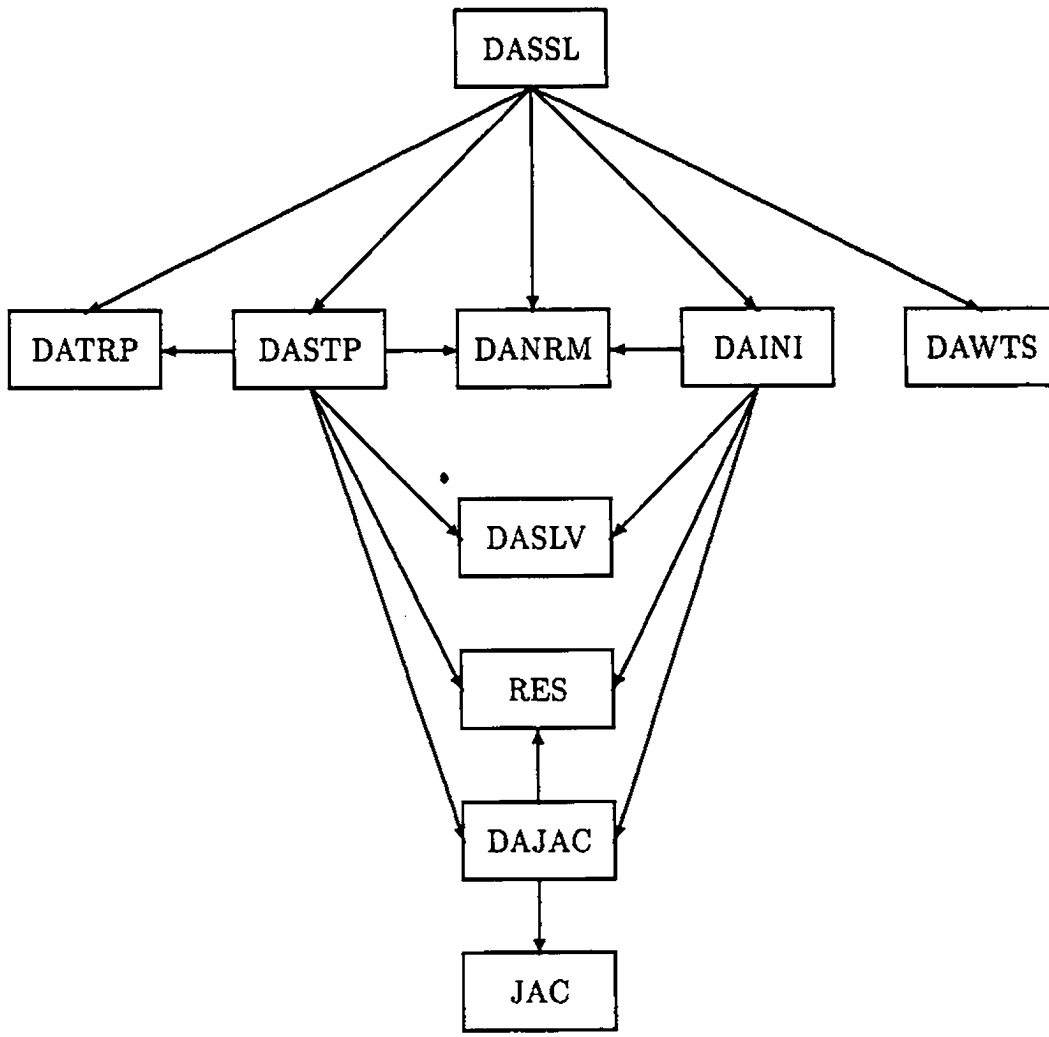

Figure 3.3: The calling tree of DASSL.

The purpose of each routine pictured in Fig. 3.3 is as follows

- DASSL - Driver routine.
- DATRP - To use polynomials to approximate the solution.
- DASTP - One-step solver to advance the solution by one time step.
- DANRM - To compute the weighted root-mean-square norm of a vector.
- DAINI - To find the initial YPRIME.
- DAWTS - To set the error weight vector.
- DASLV - To solve the linear system.
- RES - To define the system of DAEs (User defined).
- DAJAC - To compute the iteration matrix and form the LU-decomposition.
- JAC - To define the Jacobian matrix (User defined).

The driver routine DASSL allocates storage, and checks for illegal input and other error conditions. It also sets up the initial step size and optionally calls the DAINI subroutine to compute the initial derivative. If the user provides the initial solution and derivative values for the ODE system, they must be chosen to be consistent. For our case, the consistency of the initial conditions is guaranteed by Algorithm 3.2. DASSL gives the user two options for the linear algebra solver. They are the LINPACK [21] routines GEFA, for a full Jacobian matrix, and GBFA, for a banded Jacobian matrix.

The call to DASSL is

```
      CALL DASSL(RES,NEQ,T,Y,YPRIME,TOUT,INFO,RTOL,ATOL,
     *           IDID,RWORK,LRW,IWORK,LIW,RPAR,IPAR,JAC)
```

We will discuss only the parameters which have be modified, referring the reader to the detailed discussion of all parameters in the documentation for DASSL. As mentioned in [15], many of DASSL's features can be activated by setting an element in the array INFO. We have added one element to this array: INFO(16) = 1 corresponds to our modification and INFO(16) = 0 corresponds to using DASSL without our modification. In the original DASSL, the parameters RPAR and IPAR are provided for communication between the DASSL and the subroutine RES. We use RPAR to store the values of the Jacobian matrix, the eigenvalues and eigenvectors for matrix decomposition, etc, and IPAR to store the information about the matrix structure, such as the number of collocation points and the number of intervals, etc.

In summary, the modifications to DASSL are

- the addition of an option to the routine DAJAC to factor matrix (3.20) by a sequence of calls to ARCEDC.
- the addition of a similar option to the routine DASLV to make a sequence of calls to ARCESL.

Here ARCEDC is the factorization routine of the ABD solver, ARCECO, and ARCESL is the back substitution routine of ARCECO.

3.4 Time Dependent Coefficients

In the proceeding discussion L_1, L_2 were assumed to be independent of time. If either L_1 or L_2 is dependent on time, then changes must be made to the algorithm. There are four distinct cases, namely,

(i) L_1 and L_2 are both independent of t,

(ii) L_1 is dependent on t, L_2 is independent of t,

(iii) L_1 is independent of t, L_2 is dependent on t,

(iv) L_1 and L_2 are both dependent on t.

For the moment let us assume that both operators depend on time (case (iv)). Therefore we consider the following linear parabolic equation with homogeneous Dirichlet boundary conditions,

$$\frac{\partial u}{\partial t} = (L_{1,t} + L_{2,t})u + f(x_1, x_2, t), \quad (x_1, x_2, t) \in \Omega \times [0, T], \tag{3.25}$$

$$u(x_1, x_2, t) = 0, \quad (x_1, x_2) \in \partial\Omega, \tag{3.26}$$

$$u(x_1, x_2, 0) = \theta(x_1, x_2), \quad (x_1, x_2) \in \Omega, \tag{3.27}$$

where

$$L_{1,t} = a_1(x_1, t)\frac{\partial^2}{\partial x_1^2} + c_1(x_1, t), \quad a_1(x_1, t) > 0, \tag{3.28}$$

$$L_{2,t} = a_2(x_2, t)\frac{\partial^2}{\partial x_2^2} + b_2(x_2, t)\frac{\partial}{\partial x_2} + c_2(x_2, t), \quad a_2(x_2, t) > 0, \tag{3.29}$$

where $\Omega = [0, 1] \times [0, 1]$. Using orthogonal spline collocation, we require the approximation $U \in \mathcal{M}_1(r_1, \pi_1) \otimes \mathcal{M}_2(r_2, \pi_2)$ to satisfy equation (3.25) at the Gaussian points. We obtain the following system of ODEs,

$$(B_1 \otimes B_2)\frac{d\vec{u}}{dt} = (A_1(t) \otimes B_2 + B_1 \otimes A_2(t))\vec{u} + \vec{f}(t), \; t \in [0, T], \tag{3.30}$$

where

$$A_i = (a_{m\,n}^{(i)}(t))_{m,n=1}^{M_i}, \quad a_{mn}^{(i)}(t) = L_{i,t}\phi_n^{(i)}(\xi_i^{(m)}, t), \; i = 1, 2,$$

B_i, $i = 1, 2$, are defined as in (2.12) and $\vec{u}$, $\vec{f}(t)$ are defined as in (3.8), (3.9), respectively. In the previous case, (3.1)-(3.5), the matrices A_1, B_1 were time independent, and hence it was possible to apply the matrix tensor product decomposition to the ODE system prior to providing it to DASSL. Here we defer the matrix tensor product decomposition until a later point inside DASSL. When equation (3.30) is given to DASSL, the Jacobian matrix required for each time iteration will be,

$$(A_1(t) - c * B_1) \otimes B_2 + B_1 \otimes A_2(t),$$

where c, as before, is a scalar chosen by DASSL.

For each time step, we need to solve a linear system involving the Jacobian of the form,

$$((A_1(t) - c * B_1) \otimes B_2 + B_1 \otimes A_2(t))\vec{x} = \vec{r}. \tag{3.31}$$

We can now apply the matrix decomposition approach at this stage. For a given t, it follows from Lemma 2.1 and Lemma 2.2 that there exist a diagonal matrix $\Lambda(t)$ and a matrix $Z(t)$ such that

$$Z^T(t)G_1(t)Z(t) = \Lambda(t), \quad Z^T(t)F_1(t)Z(t) = I_1, \tag{3.32}$$

where $G_1(t)$ and $F_1(t)$ are defined as $G_1(t) = B_1^T W D(t) A_1(t)$, $F_1(t) = B_1^T W D(t) B_1$, with $W = diag(h_1^{(1)}w_1^{(1)}, \cdots, h_1^{(1)}w_1^{(r_1-1)}, \cdots, h_1^{(N_1)}w_1^{(1)}, \cdots, h_1^{(N_1)}w_1^{(r_1-1)})$, and $D(t) = diag(1/a_1(\xi_1^{(1)}, t), \cdots, 1/a_1(\xi_1^{(M_1)}, t))$. As a result, from (3.31) and (3.32) we get,

$$((Z^T(t)B_1^T W D(t)) \otimes I_2)(A_1(t) \otimes B_2 + B_1 \otimes A_2(t) - c * B_1 \otimes B_2)$$
$$(Z^T(t) \otimes I_2)(Z^{-1}(t) \otimes I_2)\vec{x} = ((Z^T(t)B_1^T W D(t)) \otimes I_2)\vec{r}$$

which is equivalent to

$$((\Lambda(t) - c * I) \otimes B_2 + I_1 \otimes A_2(t))\vec{v} = \vec{g}, \tag{3.33}$$

where $\vec{v} = (Z^{-1}(t) \otimes I_2)\vec{x}$ and $\vec{g} = ((Z^T(t)B_1^T W D(t)) \otimes I_2)\vec{r}$.

We have modified DASSL to use the following algorithm to solve (3.31) at each time step:

Step 1 Determine $\Lambda(t)$ and $Z(t)$ satisfying
$Z(t)^T G_1(t) Z(t) = \Lambda(t), \quad Z(t)^T F_1(t) Z(t) = I_1.$

Step 2 Compute $\vec{g} = (Z^T(t)B_1^T W D(t) \otimes I_2)\vec{r}$.

Step 3 Solve $((\Lambda(t) - c * I) \otimes B_2 + I_1 \otimes A_2(t))\vec{v} = \vec{g}$.

Step 4 Compute $\vec{x} = (Z(t) \otimes I_2)\vec{v}$.

Algorithm 3.3

Next we given the procedure for forming the collocation matrices $A_1(t)$ and $A_2(t)$. Let B_i' and B_i'', $i = 1, 2$, be matrices such that,

$$B_i' = (b_{m\,n}^{\prime(i)})_{m,n=1}^{M_i}, \quad b_{mn}^{\prime(i)} = [\phi_n^{(i)}]'(\xi_i^{(m)}) \tag{3.34}$$

$$B_i'' = (b_{m\,n}^{\prime\prime(i)})_{m,n=1}^{M_i}, \quad b_{mn}^{\prime\prime(i)} = [\phi_n^{(i)}]''(\xi_i^{(m)}) \;\; i = 1,\, 2. \tag{3.35}$$

Let C_i and C_i'' be matrices such that,

$$C_i = (c_{m\,n}^{(i)})_{m,n=1}^{M_i}, \quad c_{mn}^{(i)} = c_i(\xi_i^{(m)}, t). \tag{3.36}$$

$$C_i'' = (c_{m\,n}^{\prime\prime(i)})_{m,n=1}^{M_i}, \quad c_{mn}^{\prime\prime(i)} = a_i(\xi_i^{(m)}, t), \;\; i = 1, 2. \tag{3.37}$$

Let C_2' be matrix such that,

$$C_2' = (c_{m\,n}^{\prime(2)})_{m,n=1}^{M_2}, \quad c_{mn}^{\prime(2)} = b_2(\xi_2^{(m)}, t). \tag{3.38}$$

Each entry of the matrices C_i, C_i'' and C_2' is a function of t. In order to compute the matrices $A_1(t)$ and $A_2(t)$ efficiently, we use the equations,

$$A_1(t) = C_1'' \star B_1'' + C_1 \star B_1, \tag{3.39}$$

$$A_2(t) = C_2'' \star B_2'' + C_2' \star B_2' + C_2 \star B_2, \tag{3.40}$$

where $\star$ is the Hadamard Product. Therefore we can compute the time independent matrices by calling the B-splines package and at each time iteration, for a given t, we can form the matrices $A_1(t)$ and $A_2(t)$ by equations (3.39) and (3.40). It is easy to see that this step can be done in parallel.

The two algorithms which have described, one for the time independent coefficient case and one for the time dependent coefficient case differ in two principal ways. In the time independent case, the matrix decomposition is carried out on the ODEs themselves whereas in the time dependent case it is carried out only at the level of solving the linear systems. A consequence of this is that Step 4 in Algorithm 3.1 is carried out at the end point T, while Step 4 of Algorithm 3.3 is done at every time step. The second principal difference between the two algorithms is that for time dependent L_1, G_1 and F_1 are computed and the generalized eigenvalue problem is

solved for every time step when the Jacobian matrix is evaluated. For time dependent L_2, the matrix $A_2(t)$, must be recomputed at every time step. Some savings can be made in the cases (ii), (iii). For case (ii), A_2 is computed only once, but the generalized eigenvalue problem must be solved for every time step. For case (iii), the eigenvalues and eigenvectors are computed only once, but $A_2(t)$ must be recomputed at every time step.

In our code these four cases are allowed by 2 switches, TDEPCX and TDEPCY. Set TDEPCX = 0 if L_1 is time dependent and set TDEPCX $\neq$ 0 if L_1 is time independent. Similarly, set TDEPCY = 0 if L_2 is time dependent and set TDEPCX $\neq$ 0 if L_2 is time independent.

3.5 Performance Analysis and Numerical Experiments

In order to demonstrate the parallel performance of our algorithms, we solved the test problems listed in Table 3.1. The experiments were carried out with M = 100, k = 2, and the user defined parameters for DASSL taken as TOUT = 1, RTOL = 0 and ATOL = 10^{-10}. Here M is the number of subintervals, k is the number of collocation points in each subinterval, TOUT is the time value at which a solution is desired, ATOL is the absolute tolerance used by DASSL and RTOL = 0 implies a pure absolute error tolerance is being applied. The profiling was carried out on an Alliant/FX2800.

(3.1)	$L_1 = (x_1^2+1)\frac{\partial^2}{\partial x_1^2} + x_1,$
	$L_2 = (x_2^2+1)\frac{\partial^2}{\partial x_2^2} + x_2\frac{\partial}{\partial x_2} + x_2,$
	True solution $u = (e^{-t}+1)\sin(\pi x_1)\sin(\pi x_2)$
(3.2)	$L_1 = (t+1)(x_1^2+2)\frac{\partial^2}{\partial x_1^2} + t\sin(\pi x_1),$
	$L_2 = (t+2)(x_2^4+5)\frac{\partial^2}{\partial x_2^2} + t\sin(\pi x_2)\frac{\partial}{\partial x_2} + \cos(\pi x_2)t^2,$
	True solution $u = (e^{-t}+1)\sin(\pi x_1)\sin(\pi x_2)$
(3.3)	$L_1 = (1+t+x_1)\frac{\partial^2}{\partial x_1^2} + t,$
	$L_2 = (1+x_2^2)\frac{\partial^2}{\partial x_2^2} + x_2\frac{\partial}{\partial x_2} + x_2,$
	True solution $u = (e^{-t}+1)\sin(\pi x_1)\sin(\pi x_2)$

Table 3.1: Test problems for the parabolic solver.

3.5.1 Time Profiling and Speedup

In a manner similar to that of Chapter 2, we concentrate on the routines which can be executed through a sequence of independent calls. These routines are ZTATV, ZV, ARCE, EIGENV, ARCEDC, ARCESL and FUNRH. Routines ZTATV, ZV, ARCE and EIGENV have been described in Chapter 2. The roles of routines ARCEDC, ARCESL and FUNRH are as follows,

- ARCEDC performs the modified alternate row and column decomposition, with partial pivoting, of an almost block diagonal matrix. This routine is from the package ARCECO [20].
- ARCESL finds the solution of the linear system $A\vec{x} = \vec{b}$ using the decomposition of the matrix A already generated in ARCEDC. This routine is also from the package ARCECO.

- FUNRH evaluates the right hand side function of the partial differential equation, ie the function f in equation (3.1).

There are some other routines which also can be embedded in a sequence of independent calls, such as FUNINT, which evaluates the initial function of the partial differential equation, but the running times of these routines make up a very small fraction of total running time, and are not significant when we calculate the total fraction of parallelizable code. On the other hand, the routine DANRM can not be embedded in a sequence of independent calls, but does make a significant contribution to the overall run time. However, it is possible to parallelize DANRM at a lower level. If we have a vector $\vec{v}$ with length n, then DANRM returns the weighted root-mean-square norm of the vector $\vec{v}$,

$$||\vec{v}|| = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{v(i)}{w(i)}\right)^2}.$$

We can perform the calculations of $(v(i)/w(i))^2$, $i = 1, \cdots, n$, in parallel and thus significant improve the parallel efficiency of DANRM. The time which we report for DANRM is the time spent on calculating $(v(i)/w(i))^2$, $i = 1, \cdots, n$; this represents most of the execution time for DANRM.

In the following table we present the results of profiling of our code, which we call PARCOL, for the test problems given in Table 3.1. Time is given in seconds. We also provide the overall running time for PARCOL.

N_P	Routine	Problem 3.1		Problem 3.2		Problem 3.3	
		CPU_T	%	CPU_T	%	CPU_T	%
1	EIGENV	9.5	0.33	367.9	7.78	256.1	6.82
	ZTATV	918.6	32.41	1432.0	30.28	1139.5	30.35
	ZV	874.3	30.85	1342.7	28.39	1072.7	28.56
	ARCEDC	34.9	1.23	51.5	1.08	35.8	0.95
	ARCESL	130.6	4.61	202.5	4.28	161.6	4.30
	ALBDVE	262.9	9.28	407.7	8.62	325.3	8.66
	FUNRH	286.4	10.10	397.0	8.39	340.3	9.06
	DANRM	88.5	3.12	129.4	2.73	105.0	2.79
	PARCOL	2833.7		4728.5		3754.7	
4	EIGENV	3.0	0.29	117.0	6.67	81.0	5.93
	ZTATV	211.0	20.96	330.5	18.85	258.8	18.95
	ZV	229.1	22.76	357.2	20.37	278.4	20.38
	ARCEDC	9.9	0.98	15.0	0.85	10.3	0.75
	ARCESL	36.2	3.60	56.4	3.22	44.7	3.27
	ALBDVE	69.4	6.89	108.4	6.18	85.4	6.25
	FUNRH	83.5	8.30	116.9	6.67	101.0	7.39
	DANRM	25.1	2.50	36.9	2.10	29.7	2.17
	PARCOL	1006.3		1753.4		1365.9	
	To be continued in next page						

8	EIGENV	1.5	0.22	59.3	4.98	40.9	4.40
	ZTATV	120.1	17.35	181.4	15.2	142.6	15.34
	ZV	131.0	18.92	198.2	16.6	155.4	16.71
	ARCEDC	5.2	0.76	7.8	0.65	5.3	0.57
	ARCESL	18.5	2.67	28.5	2.40	22.7	2.45
	ALBDVE	35.9	5.19	55.4	4.66	43.9	4.72
	FUNRH	42.5	6.14	58.7	4.94	50.7	5.45
	DANRM	12.7	1.83	18.7	1.57	15.1	1.62
	PARCOL	692.1		1189.6		929.7	

Table 3.2: Profiling for problems 3.1-3.3.

Table 3.2 shows that the running time of the listed routines decreases in a linear fashion as the number of processor increases. To determine the maximum speedup, we need to calculate the the percentage of time spend in the parallelizable code when run on a single processor. This value is computed by adding the time spent on EIGENV, ZTATV, ZV, ARCEDC, ARCESL, FUNRH, and part of DANRM and dividing by the total time. In Table 3.3, we show the actual speedup we obtained and the maximum speedup we can expect. The maximum speedup is calculated by Amdahl's law (1.19) with $n = 8$ and the actual speedup is computed by (1.18) given in section 1.2.4.

N_P	Problem 3.1	Problem 3.2	Problem 3.3
4	2.81	2.69	2.74
8	4.09	3.97	4.03
Max. Speedup	5.11	5.02	5.01

Table 3.3: Speedup of the test problems 3.1-3.3 on an Alliant/FX2800.

3.5.2 Convergence

In the following, we show the convergence properties of our algorithm. The results here are experimental since they are only based on the observation on the output of numerical experiments.

Table 3.4 gives the maximum error at mesh points for the test problems 3.1-3.3 for the number of collocation points $k = 2$ and 3, and for the number of intervals M= 4, 8, 16, 32, and 64. Here we choose the same numbers of collocation points in each subinterval along the x_1 and x_2 directions. In Table 3.4 we also report the values R which are computed by (1.13) for consecutive step sizes.

	Problem 3.1		Problem 3.2		Problem 3.3	
	Errors.					
M	$k=2$	$k=3$	$k=2$	$k=3$	$k=2$	$k=3$
4	9.6-4	4.9-7	9.6-3	6.8-7	9.7-4	4.6-7
8	6.0-5	9.8-9	5.9-5	1.2-8	6.0-5	1.1-8
16	3.7-6	1.5-10	3.7-6	1.8-10	3.7-6	1.9-10
32	2.3-7	1.6-11	2.3-7	3.0-11	2.3-7	1.8-11
64	1.4-8	4.8-11	1.4-8	2.7-11	1.4-8	2.5-11
	The ratios computed by (1.13).					
8	4.00	5.64	7.34	5.82	4.01	5.38
16	4.01	6.02	3.99	6.05	4.01	5.85
32	4.00	3.22	4.00	2.58	4.00	3.39
64	4.03	-1.58	4.03	0.15	4.03	-0.47

Table 3.4: The error at mesh points for test problems 3.1-3.3

From Table 3.4 we can see that the convergence of order $O(h^{2k})$ has been achieved whenever the error is bigger than the tolerance of the time integration, here chosen as 10^{-10}. These results support the conjecture that the error bound of this algorithm

at mesh points is $O(h^{2k}) + O(TOL)$, where TOL is the error tolerance of the time integration.

3.6 On More General Boundary Conditions

For constants α, β, γ, δ, let the functional operators $V_{\alpha,\beta}$ and $H_{\gamma,\delta}$ be defined as (2.27) and (2.28) respectively.

We consider (3.1) subject to the following time dependent boundary conditions

$$V_{\alpha_0,\beta_0}u(0,x_2,t) \equiv g_0(x_2,t), \tag{3.41}$$

$$V_{\alpha_1,\beta_1}u(1,x_2,t) \equiv g_1(x_2,t), \tag{3.42}$$

$$H_{\gamma_0,\delta_0}u(x_1,0,t) \equiv h_0(x_1,t), \tag{3.43}$$

$$H_{\gamma_1,\delta_1}u(x_1,1,t) \equiv h_1(x_1,t). \tag{3.44}$$

Let $\hat{\mathcal{M}}_i(r_i,\pi_i)$, $i=1,2$, be defined as in (2.37) and $\{\phi_{n_i}^{(i)}(x)\}_{n_i=0}^{M_i+1}$ be a B-spline type basis for $\hat{\mathcal{M}}_i(r,\pi_i)$ satisfying (2.38)-(2.39). We write the orthogonal spline collocation solution $U \in \hat{\mathcal{M}}_1(r_1,\pi_1)\otimes\hat{\mathcal{M}}_2(r_2,\pi_2)$ as

$$U(x_1,x_2,t) = \sum_{n_1=0}^{M_1+1}\sum_{n_2=0}^{M_2+1} u_{n_1,n_2}(t)\phi_{n_1}^{(1)}(x_1)\phi_{n_2}^{(2)}(x_2). \tag{3.45}$$

Collocating at M_1M_2 internal Gauss points of Ω, we get the following M_1M_2 equations,

$$\frac{\partial U}{\partial t} = (L_1+L_2)U(\xi_1^{(m_1)},\xi_2^{(m_2)},t) + f(\xi_1^{(m_1)},\xi_2^{(m_2)},t), \tag{3.46}$$

for $m_i = 1,\ldots,M_i$, $i=1,2$. Note that we consider the case with time independent coefficients, the time dependent case can be done in a similar way.

Similar to the result in Chapter 2, we can get that,

$$u_{0,0,}(t) = \alpha_0 h_0(0,t) - \beta_0 h_{0,x_2}(0,t),$$

$$u_{0,M_2+1}(t) = \gamma_1 g_0(1,t) - \delta_1 g_{0,x_1}(1,t),$$

$$u_{M_1+1,0}(t) = \gamma_0 g_1(0,t) - \delta_0 g_{1,x_1}(0,t),$$

$$u_{M_1+1,M_2+1}(t) = \alpha_1 h_1(1,t) - \beta_1 h_{1,x_2}(1,t).$$

Based on the above results and the interpolation at the Gaussian points on the boundary, we have

$$\sum_{n_2=1}^{M_2} u_{0,n_2}(t)\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) = g_0(\xi_2^{(m_2)},t) - u_{0,0}(t)\phi_0^{(2)}(\xi_2^{(m_2)}) - u_{0,M_2+1}(t)\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}), \tag{3.47}$$

$$\sum_{n_2=1}^{M_2} u_{M_1+1,n_2}(t)\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) = g_1(\xi_2^{(m_2)},t) - u_{M_1+1,0}(t)\phi_0^{(2)}(\xi_2^{(m_2)}) - u_{M_1+1,M_2+1}(t)\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}), \tag{3.48}$$

$$\sum_{n_1=1}^{M_1} u_{n_1,0}(t)\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) = h_0(\xi_1^{(m_1)},t) - u_{0,0}(t)\phi_0^{(1)}(\xi_1^{(m_1)}) - u_{M_1+1,0}(t)\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}), \tag{3.49}$$

$$\sum_{n_1=1}^{M_1} u_{n_1,M_2+1}(t)\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) = h_1(\xi_1^{(m_1)},t) - u_{0,M_2+1}(t)\phi_0^{(1)}(\xi_1^{(m_1)}) - u_{M_1+1,M_2+1}(t)\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}). \tag{3.50}$$

We next set $\vec{u}$ and $\vec{f}$ as in (2.8) and (2.9), respectively, but with,

$$\begin{aligned}
f_{m_1,m_2}(t) &= f(\xi_1^{(m_1)},\xi_2^{(m_2)},t) - L_1\phi_0^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{0,n_2}(t)\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- \phi_0^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{0,n_2}(t)L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- L_1\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{M_1+1,n_2}(t)\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- \phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\sum_{n_2=0}^{M_2+1} u_{M_1+1,n_2}(t)L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&- L_2\phi_0^{(2)}(\xi_2^{(m_2)})\sum_{n_1=1}^{M_1} u_{n_1,0}(t)\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- \phi_0^{(2)}(\xi_2^{(m_2)})\sum_{n_1=1}^{M_1} u_{n_1,0}(t)L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) \\
&- L_2\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)})\sum_{n_1=1}^{M_1} u_{n_1,M_2+1}(t)\phi_{n_1}^{(1)}(\xi_1^{(m_1)})
\end{aligned}$$

$$- \phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1}(t) L_1 \phi^{(1)}_{n_1}(\xi_1^{(m_1)}), \tag{3.51}$$

$m_i = 1, \ldots, M_i$, $i = 1, 2$. We can simplify the right hand side of equation (3.51) by using equations (3.47)-(3.50),

$$\begin{aligned} f_{m_1,m_2}(t) &= f(\xi_1^{(m_1)}, \xi_2^{(m_2)}, t) - L_1\phi^{(1)}_0(\xi_1^{(m_1)}) g_0(\xi_2^{(m_2)}, t) \\ &- \phi^{(1)}_0(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{0,n_2}(t) L_2 \phi^{(2)}_{n_2}(\xi_2^{(m_2)}) - L_1\phi^{(1)}_{M_1+1}(\xi_1^{(m_1)}) g_1(\xi_2^{(m_2)}, t) \\ &- \phi^{(1)}_{M_1+1}(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{M_1+1,n_2}(t) L_2 \phi^{(2)}_{n_2}(\xi_2^{(m_2)}) - L_2\phi^{(2)}_0(\xi_2^{(m_2)}) h_0(\xi_1^{(m_1)}, t) \\ &- \phi^{(2)}_0(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,0}(t) L_1 \phi^{(1)}_{n_1}(\xi_1^{(m_1)}) - L_2\phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}) h_1(\xi_1^{(m_1)}, t) \\ &- \phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1}(t) L_1 \phi^{(1)}_{n_1}(\xi_1^{(m_1)}). \end{aligned} \tag{3.52}$$

The ODE system (3.46) can be reduced to the form (3.10) with $\vec{f}(t)$ defined as above, and Algorithm 3.1 can be applied.

The the initial value, $\vec{u}(0)$ can be solved from the following equation

$$(B_1 \otimes B_2)\vec{u}(0) = \vec{\theta}, \tag{3.53}$$

where

$$\begin{aligned} \theta_{m_1,m_2} &= \theta(\xi_1^{(m_1)}, \xi_2^{(m_2)}) - \phi^{(1)}_0(\xi_1^{(m_1)}) g_0(\xi_2^{(m_2)}, 0) \\ &- \phi^{(1)}_{M_1+1}(\xi_1^{(m_1)}) g_1(\xi_2^{(m_2)}, 0) - \phi^{(2)}_0(\xi_2^{(m_2)}) h_0(\xi_1^{(m_1)}, 0) \\ &- \phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}) h_1(\xi_1^{(m_1)}, 0) + u_{0,0}\phi^{(1)}_0(\xi_1^{(m_1)}) \phi^{(2)}_0(\xi_2^{(m_2)}) \\ &+ u_{M_1+1,0}\phi^{(1)}_{M_1+1}(\xi_1^{(m_1)}) \phi^{(2)}_0(\xi_2^{(m_2)}) + u_{0,M_2+1}\phi^{(1)}_0(\xi_1^{(m_1)}) \\ &+ \phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}) + u_{M_1+1,M_2+1}\phi^{(1)}_{M_1+1}(\xi_1^{(m_1)}) \phi^{(2)}_{M_2+1}(\xi_2^{(m_2)}). \end{aligned}$$

Now we define $h_i^{(t)} = \frac{\partial h_i}{\partial t}$ and $g_i^{(t)} = \frac{\partial g_i}{\partial t}$ for $i = 0, 1$. then we have

$$\begin{aligned} u'_{0,0,}(t) &= \alpha_0 h_0^{(t)}(0, t) - \beta_0 h^{(t)}_{0,x_2}(0, t), \\ u'_{0,M_2+1}(t) &= \gamma_1 g_c^{(t)}(1, t) - \delta_1 g^{(}_{0,x_i} t)(1, t), \end{aligned}$$

$$u'_{M_1+1,0}(t) = \gamma_0 g_1^{(t)}(0,\, t) - \delta_0 g_{1,x_1}^{(t)}(0,\, t),$$
$$u'_{M_1+1,\,M_2+1}(t) = \alpha_1 h_1^{(t)}(1,\, t) - \beta_1 h_{1,x_2}^{(t)}(1,\, t),$$

and

$$\sum_{n_2=1}^{M_2} \phi_{n_2}^{(2)}(\xi_2^{(m_2)}) u'_{0,n_2}(t) = g_0^{(t)}(\xi_2^{(m_2)}, t) - u'_{0,0}(t)\phi_0^{(2)}(\xi_2^{(m_2)}) - u'_{0,M_2+1}(t)\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}), \tag{3.54}$$

$$\sum_{n_2=1}^{M_2} \phi_{n_2}^{(2)}(\xi_2^{(m_2)}) u'_{M_1+1,n_2}(t) = g_1^{(t)}(\xi_2^{(m_2)}, t) - u'_{M_1+1,0}(t)\phi_0^{(2)}(\xi_2^{(m_2)}) - u'_{M_1+1,M_2+1}(t)\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}), \tag{3.55}$$

$$\sum_{n_1=1}^{M_1} \phi_{n_1}^{(1)}(\xi_1^{(m_1)}) u'_{n_1,0}(t) = h_0^{(t)}(\xi_1^{(m_1)}, t) - u'_{0,0}(t)\phi_0^{(1)}(\xi_1^{(m_1)}) - u'_{M_1+1,0}(t)\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}), \tag{3.56}$$

$$\sum_{n_1=1}^{M_1} \phi_{n_1}^{(1)}(\xi_1^{(m_1)}) u'_{n_1,M_2+1}(t) = h_1^{(t)}(\xi_1^{(m_1)}, t) - u'_{0,M_2+1}(t)\phi_0^{(1)}(\xi_1^{(m_1)}) - u'_{M_1+1,M_2+1}(t)\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}). \tag{3.57}$$

The vector $\frac{d\vec{u}}{dt}(0)$ satisfies the following equation,

$$(B_1 \otimes B_2)\frac{d\vec{u}}{dt}(0) = \vec{\psi}, \tag{3.58}$$

where,

$$\begin{aligned}
\psi_{m_1,m_2} &= \sum_{n_1=1}^{M_1}\sum_{n_2=1}^{M_2} u_{n_1,n_2}[\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)}) + \phi_{n_1}^{(1)}(\xi_1^{(m_1)}) L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)})] \\
&+ f(\xi_1^{(m_1)}, \xi_2^{(m_2)}, 0) + (L_1\phi_0^{(1)}(\xi_1^{(m_1)}))g_0(\xi_2^{(m_2)}, 0) \\
&+ (L_1\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}))g_1(\xi_2^{(m_2)}, 0) + (L_2\phi_0^{(2)}(\xi_2^{(m_2)}))h_0(\xi_1^{(m_1)}, 0) \\
&+ (L_2\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}))h_1(\xi_1^{(m_1)}, 0) - \phi_0^{(1)}(\xi_1^{(m_1)})g_0^{(t)}(\xi_2^{(m_2)}, 0) \\
&- \phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})g_1^{(t)}(\xi_2^{(m_2)}, 0) - \phi_0^{(2)}(\xi_2^{(m_2)})h_0^{(t)}(\xi_1^{(m_1)}, 0) \\
&- \phi_{M_2+1}^{(2)}(\xi_2^{(m_2)})h_1^{(t)}(\xi_1^{(m_1)}, 0) + \phi_0^{(1)}(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{0,n_2} L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) \\
&+ \phi_{M_1+1}^{(1)}(\xi_1^{(m_1)}) \sum_{n_2=1}^{M_2} u_{M_1+1,n_2} L_2\phi_{n_2}^{(2)}(\xi_2^{(m_2)}) + \phi_0^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,0} L_1\phi_{n_1}^{(1)}(\xi_1^{(m_1)})
\end{aligned}$$

$$
\begin{aligned}
&+ \phi_0^{(2)}(\xi_2^{(m_2)}) \sum_{n_1=1}^{M_1} u_{n_1,M_2+1} L_1 \phi_{n_1}^{(1)}(\xi_1^{(m_1)}) + u'_{0,0}\phi_0^{(1)}(\xi_1^{(m_1)})\phi_0^{(2)}(\xi_2^{(m_2)}) \\
&+ u'_{M_1+1,0}\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\phi_0^{(2)}(\xi_2^{(m_2)}) + u'_{0,M_2+1}\phi_0^{(1)}(\xi_1^{(m_1)})\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}) \\
&+ u'_{M_1+1,M_2+1}\phi_{M_1+1}^{(1)}(\xi_1^{(m_1)})\phi_{M_2+1}^{(2)}(\xi_2^{(m_2)}).
\end{aligned}
$$

Hence the initial value $\frac{d\vec{w}}{dt}(0)$ can be computed by applying Algorithm 3.2, replacing $\vec{\theta}$ with $\vec{\psi}$ as given above.

We have implemented Algorithm 3.1 for general boundary conditions in the form of (3.41)-(3.44). To show the performance of our implementation, we solve the equation:

$$\frac{\partial u}{\partial t} = (L_1 + L_2)u + f(x_1, x_2, t), \tag{3.59}$$

where

$$L_1 = \frac{\partial^2}{\partial x_1^2} \text{ and } L_2 = \frac{\partial^2}{\partial x_2^2}.$$

The boundary conditions are

$$
\left.
\begin{aligned}
&u(0,x_2) - u_{x_1}(0,x_2) = (e^{-t}+1)(x_2^m - x_2^{m-1} + 1), \\
&u(1,x_2) = (e^{-t}+1)(2 + x_2^{m-1} + x_2^m), \\
&u(x_1,0) - u_{x_2}(x_1,0) = (e^{-t}+1)(x_1^m + 1), \\
&u(x_1,1) = (e^{-t}+1)(x_1^m + x_1 + 2).
\end{aligned}
\right\} \tag{3.60}
$$

The initial condition is

$$u(x_1, x_2, 0) = 2(x_1^m + x_2^m + x_1 x_2^{m-1} + 1). \tag{3.61}$$

The function f is chosen such that the solution is $(e^{-t}+1)(x_1^m + x_2^m + x_1 x_2^{m-1} + 1)$. We solve the equation for $m = 6$. In Table 3.5 we report the L_∞ error and the maximum error at mesh points for various values of M and k where M is the number of intervals and k is the number of collocation points in each subinterval. Again we use E_u to denote the L_∞ error and E_m to denote the maximum error at mesh points. R_m and R_u are computed by (1.13) for consecutive step sizes. We choose the absolute error tolerance for the time integration as 10^{-10}. The computation was carried out on a Sparc 1000 work station.

From Table 3.5 we can see that the convergence of order $O(h^{k+2})$ has been achieved for both the maximum error at the mesh points and the L_∞ error. This agrees with our conjecture that the error bound for both the maximum error at the mesh points and the L_∞ error is $O(h^{k+2}) + O(TOL)$, where TOL is the error tolerance of the time integration. The degradation of the order of convergence for $M = 64$ and $k = 3$ is because that the absolute error tolerance for DASSL, 10^{-10}, is bigger than the approximate error.

k	M	E_m	R_m	E_u	R_u
2	4	1.64-03		7.52-03	
2	8	1.34-04	3.61	5.23-04	3.84
2	16	9.54-06	3.81	3.44-05	3.92
2	32	6.44-07	3.88	1.89-06	4.18
2	64	4.26-08	3.91	1.26-07	3.90
3	4	7.98-06		1.36-04	
3	8	2.05-07	5.28	4.37-06	4.95
3	16	5.73-09	5.16	1.40-07	4.96
3	32	1.68-10	5.09	4.55-09	4.94
3	64	1.11-11	3.91	1.48-10	4.94

Table 3.5: The errors in solving Equation (3.59) with boundary conditions (3.60) and initial condition (3.61) using PARCOL.

3.7 Concluding Remarks

In this chapter, we developed and implemented an algorithm for the numerical solution of a family of linear separable two space dimension parabolic PDEs using a method-of-lines approach. By collocating the elliptic operator at Gaussian points, a system of ODEs is introduced. The differential/algebraic solver DASSL is employed

to solve the ODE system. An almost block diagonal linear system solver ARCECO [20] is added to DASSL to efficiently handle the special structure of the Jacobian matrix required by DASSL. The parallel performance of this algorithm was investigated through numerical experiments on a multiple processor system, the Alliant/FX2800.

Chapter 4

Software

In this chapter, we describe the software the packages ELLDCM and PARCOL, for which the algorithms have been described in Chapter 2 and 3, respectively. Each of the packages consists of a set of independent modules. Both ELLDCM and PARCOL are written in FORTRAN 77 and use DOUBLE PRECISION real variables throughout. In Appendix A we give sample driving programs for solving test problem 2.1 in Table 2.1 and test problem 3.2 in Table 3.1, using ELLDCM and PARCOL, respectively. In this chapter, we use x to denote x_1 and y to denote x_2.

4.1 Program ELLDCM

4.1.1 The Calling Diagram

ELLDCM is designed to compute the coefficients $u_{i,j}$ of the approximate solution U, (2.6), to equation (2.1) with Dirichlet or mixed boundary conditions with a variable mesh, as described in section 2.2. Figure 4.1 shows the calling diagram for ELLDCM. The elliptic problem is completely defined by the right side function f, the coefficient functions of the operators L_1 and L_2 of (2.1), and by the boundary conditions. They are provided to ELLDCM through user supplied subroutines indicated by the circles. The right hand side function f must be provided through the FUN routine, the

coefficient functions for the operator L_1, $a_1(x)$, and $c_1(x)$, are provided through the AX and CX functions, and the coefficient functions for the operator L_2, $a_2(y)$, $b_2(y)$, and $c_2(y)$, are provided through the AY, BY, and CY functions. The boundary conditions are provided through the H0, H1, G0, and G1 functions. The routines indicated by rectangles are part of the ELLDCM package. The routines indicated dashed rectangles are routines from ARCECO, EISPACK, LINPACK, NAG, etc. The BLAS routines, [22], [23], [44], [45], are used in various places through ELLDCM but are not shown in Figure 4.1 in order to simplify the diagram.

A main program is needed in order to call ELLDCM. It serves five purposes as follows:

- Allocate the proper real and integer work storage.
- Read in the inputs.
- Define the mesh.
- Call ELLDCM.
- Evaluate the solution at required points and produce the output.

To compute the coefficients $u_{0,0}$, u_{0,M_2+1} $u_{M_1+1,0}$ and u_{M_1+1,M_2+1}, we need to know the values $g_0'(0)$, $h_1'(0)$, $h_0'(1)$ and $g_1'(1)$. These values can be either supplied by the user through the parameter list to ELLDCM or approximated by finite difference techniques. However it should be noted that the less accurate finite difference approximation will degrade the order of convergence of the approximate solution. The values defined by (2.82)-(2.88) are also returned by ELLDCM. These values are used to define the basis functions.

▭ – ELLDCM routines.
○ – User defined routines.
⬚ – Routines from other packages.

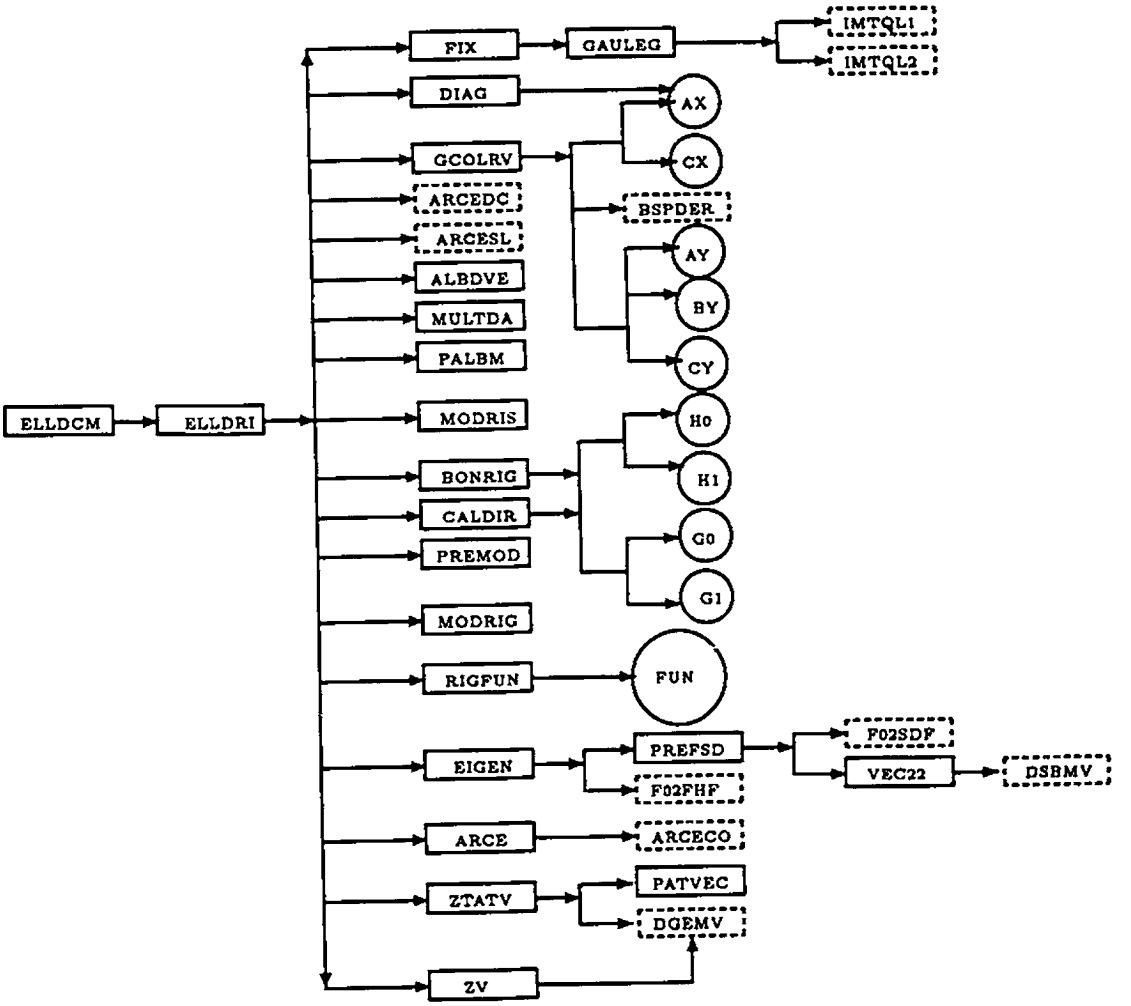

Figure 4.1: The calling tree of ELLDCM.

The ELLDCM code requires a double precision work array and an integer work array. The required amount of double precision storage is no less than,

$$58 + 30k_1N_1 + k_1^2N_1^2 + 15k_2N_2 + 10k_1^2N_1 + 3k_2^2N_2 + (81 + 17k_1)k_1/2 + (7 + k_2)k_2/2,$$

while the required integer storage is no less than,

$$3N_1 + 3N_2 + k_1N_1 + k_2N_2.$$

Also ELLDCM needs one double precision array of length $(k_1N_1 + 2)(k_2N_2 + 2)$ to store the coefficients of the solution, where k_1 and k_2 are the numbers of collocation points in each subinterval and N_1 and N_2 are the numbers of subintervals along the x and y directions respectively. The program EVALSN, which is called in order to evaluate the solution after the coefficients u_{ij}, $i = 1, \cdots, M_1$, $j = 1, \cdots, M_2$, are obtained, is discussed at the end of this chapter.

4.1.2 Using ELLDCM

In this section we give the comment section for ELLDCM.

```
      SUBROUTINE ELLDCM(KX, NBLOCX, KY, NBLOCY, AX, CX, AY, BY, CY,
     1   FUN, AO, BO, A1, B1, RO, DO, R1, D1, HO, H1, GO, G1, BFLAG,
     2   DGODYO, DH1DXO, DHODX1, DG1DY1, COEFF, NCOEFF, XI, YI,
     3   CONTX, CONTY, WORK, NWORK, IWORK, NIWORK)
C
C***********************************************************************
C
C     PURPOSE
C
C     THIS PROGRAM SOLVES ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS
C     OF THE FORM
C
C          (  L  + L  ) U = F(X,Y)
C             1    2
C
C     THE BOUNDARY CONDITIONS ARE
C
C             AO U(0,Y) - BO U_X (0,Y) = GO(Y),
C             A1 U(1,Y) - B1 U_X (1,Y) = G1(Y),
C             RO U(X,0) - DO U_Y (X,0) = HO(X),
C             R1 U(X,1) - D1 U_Y (X,1) = H1(X),
C
```

```
C     THE DOMAIN OF THE EQUATION IS OMEGA = [0,1]*[0,1]
C
C                                2                              2
C     WHERE L = -AX(X) D  + CX(X) AND L = -AY(Y) D  + BY(Y) D  + CY(Y).
C           1          X                 2          Y         Y
C
C     AX(X) > 0 FOR X IN [0,1] AND AY(Y) > 0 FOR Y IN [0,1]
C
C     THE PROGRAM ALSO RETURNS THE SCALAR VALUES IN EQUATIONS
C     (2.82)-(2.89) WHICH ARE USED FOR DEFINING THE BASIS FUNCTIONS.
C
C***********************************************************************
C
C                  ******** INPUT TO ELLDCM ********
C
C     *** ON ENTRY ***
C
C         KX         - INTEGER
C                      THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                      IN THE X DIRECTION.
C
C         NBLOCX     - INTEGER
C                      THE NUMBER OF SUBINTERVALS IN THE X DIRECTION.
C
C         KY         - INTEGER
C                      THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                      IN THE Y DIRECTION.
C
C         NBLOCY     - INTEGER
C                      THE NUMBER OF SUBINTERVALS IN THE Y DIRECTION.
C
C         AO, BO     - DOUBLE PRECISION VARIABLES
C                      THE CONSTANTS TO DEFINE THE LEFT SIDE BOUNDARY
C                      CONDITIONS.
C
C         A1, B1     - DOUBLE PRECISION VARIABLES
C                      THE CONSTANTS TO DEFINE THE RIGHT SIDE BOUNDARY
C                      CONDITIONS.
C
C         RO, DO     - DOUBLE PRECISION VARIABLES
C                      THE CONSTANTS TO DEFINE THE BOTTOM SIDE BOUNDARY
C                      CONDITIONS.
C
C         R1, D1     - DOUBLE PRECISION VARIABLES
C                      THE CONSTANTS TO DEFINE THE TOP SIDE BOUNDARY
C                      CONDITIONS.
C
C         BFLAG      - INTEGER
C                      IF THE USER CAN PROVIDE THE VALUES GO'(0),
C                      H1'(0), G1'(1) AND HO'(1), SET BFLAG = 1.
C                      OTHERWISE SET BFLAG = 0 AND THE VALUES GO'(0),
C                      H1'(1), H1'(0) AND HO'(1) WILL THEN BE
C                      APPROXIMATED BY FINITE DIFFERENCES.
```

```
C
C         DGODYO    - DOUBLE PRECISION VARIABLE
C                     DGODYO = GO'(O) WHEN BFLAG = 1. WHEN BFLAG = O,
C                     IGNORE THIS PARAMETER BY TREATING IT AS A DUMMY
C                     ARGUMENT.
C
C         DH1DXO    - DOUBLE PRECISION VARIABLE
C                     DH1DXO = H1'(O) WHEN BFLAG = 1. WHEN BFLAG = O,
C                     IGNORE THIS PARAMETER BY TREATING IT AS A DUMMY
C                     ARGUMENT.
C
C         DHODX1    - DOUBLE PRECISION VARIABLE
C                     DHODX1 = HO'(1) WHEN BFLAG = 1. WHEN BFLAG = O,
C                     IGNORE THIS PARAMETER BY TREATING IT AS A DUMMY
C                     ARGUMENT.
C
C         DG1DY1    - DOUBLE PRECISION VARIABLE
C                     DG1DY1 = G1'(1) WHEN BFLAG = 1. WHEN BFLAG = O,
C                     IGNORE THIS PARAMETER BY TREATING IT AS A DUMMY
C                     ARGUMENT.
C
C         XI        - DOUBLE PRECISION ARRAY OF LENGTH NBLOCX+1
C                     THE MESH IN THE X DIRECTION.
C
C         YI        - DOUBLE PRECISION ARRAY OF LENGTH NBLOCY+1
C                     THE MESH IN THE Y DIRECTION.
C
C         WORK      - DOUBLE PRECISION ARRAY OF LENGTH NWORK.
C                     WORK STORAGE.
C
C         NWORK     - INTEGER
C                     THE LENGTH OF WORK
C                     NWORK >= 58 + 30 KX NBLOCX + KX**2 NBLOCX**2 +
C                              15 KY NBLOCY + 10 KX**2 NBLOCX +
C                              3 KY**2 NBLOCY + ((81+17KX) KX)/2 +
C                              ((7+KY) KY)/2
C
C         IWORK     - INTEGER ARRAY OF LENGTH NIWORK.
C                     INTEGER WORK STORAGE.
C
C         NIWORK    - INTEGER
C                     THE LENGTH OF IWORK
C                     NIWORK >= 3 NBLOCX + 3 NBLOCY + KX NBLOCX +
C                               KY NBLOCY
C
C*************************************************************************
C
C                   ******** OUTPUT FROM ELLDCM ********
C
C         COEFF     - DOUBLE PRECISION ARRAY OF LENGTH NCOEFF.
C                     THE COEFFICIENTS OF THE B-SPLINES IN THE
C                     APPROXIMATE SOLUTION.
C
```

```
C          NCOEFF     - INTEGER
C                       THE LENGTH OF COEFF
C                       NCOEFF = (KX*NBLOCX+2)*(KY*NBLOCY+2).
C
C          CONTX      - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                       THE SCALAR CONSTANTS GIVEN BY (2.82)-(2.85)
C                       FOR DEFINING THE BASIS FUNCTIONS.
C
C          CONTY      - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                       THE SCALAR CONSTANTS GIVEN BY (2.86)-(2.89)
C                       FOR DEFINING THE BASIS FUNCTIONS.
C
C*********************************************   **********************
C
C                    ******** USER DEFINED FU  "   NS ***********
C
C       THE FOLLOWING USER DEFINED FUNCTIONS SH  . BE DECLARED AS
C       EXTERNAL IN THE MAIN PROGRAM. THE ARGUMEN  TO EACH FUNCTION
C       ARE DOUBLE PRECISION VARIABLES X AND Y.
C
C            AX         - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                         FUNCTION AX IN OPERATOR L1. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION AX(X)
C
C            CX         - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                         FUNCTION CX IN OPERATOR L1. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION CX(X)
C
C            AY         - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                         FUNCTION AY IN OPERATOR L2. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION AY(X)
C
C            BY         - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                         FUNCTION BY IN OPERATOR L2. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION BY(X)
C
C            CY         - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                         FUNCTION CY IN OPERATOR L2. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION CY(X)
C
C            GO         - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                         BOUNDARY CONDITIONS ALONG THE RIGHT HAND
C                         OF THE REGION. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION GO(X)
C
C            G1         - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                         BOUNDARY CONDITIONS ALONG THE LEFT HAND
C                         OF THE REGION. THE HEADING IS
C                         DOUBLE PRECISION FUNCTION G1(X)
C
C            HO         - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                         BOUNDARY CONDITIONS ALONG THE BOTTOM SIDE
C                         OF THE REGION. THE HEADING IS
```

```
C                    DOUBLE PRECISION FUNCTION HO(X)
C
C         H1         - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                      BOUNDARY CONDITIONS ALONG THE TOP SIDE
C                      OF THE REGION. THE HEADING IS
C                      DOUBLE PRECISION FUNCTION H1(X)
C
C         FUN        - DOUBLE PRECISION FUNCTION TO EVALUATE THE RIGHT
C                      HAND SIDE FUNCTION OF THE DIFFERENTIAL EQUATION.
C                      THE HEADING IS
C                      DOUBLE PRECISION FUNCTION FUN(X,Y)
C
C***********************************************************************
C
C                   ******** PACKAGE SUBROUTINES ***********
C
C         ELLDCM     - CHECKS STORAGE PARAMETERS, BREAKS UP THE WORK
C                      AREA. CALLS ELLDRI.
C
C         ELLDRI     - MANAGES ALL CALLS TO PROGRAM MODULES.
C                      CALLS FIX, GCOLRV, DIAG, MULTDA, PALBM, EIGENV,
C                      BONRIG, CALDER, ARCEDC, MODRIS, ARCESL, PREMOD,
C                      ALBDVE, RIGFUN, MODRIG, ZTATV, ARCE AND ZV.
C
C         FIX        - COMPUTES THE FIXED VALUES, SUCH AS GAUSSIAN
C                      POINTS ETC. CALLS GAULEG.
C
C         GAULEG     - COMPUTES THE POINTS AND WEIGHTS FOR
C                      GAUSS-LEGENDRE QUADRATURE OVER THE INTERVAL
C                      [0,1].
C
C         DIAG       - COMPUTES THE DIAGONAL MATRIX W*D DEFINED IN
C                      LEMMA 2.1.
C
C         GCOLRV     - COLLOCATION MATRIX SETUP ROUTINE. THIS ROUTINE
C                      CALLS THE B-SPLINE ROUTINE BSPDER, IN THE
C                      MODIFIED FORM AS IN COLSYS. ALSO THIS ROUTINE
C                      RETURNS NECESSARY VALUES FOR HANDLING THE
C                      BOUNDARY CONDITIONS.
C
C         MULTDA     - COMPUTES THE VALUE OF DD*A, WHERE DD IS A
C                      DIAGONAL MATRIX AND A IS AN ALMOST BLOCK
C                      DIAGONAL MATRIX.
C
C         ALBDVE     - COMPUTES A*V, WHERE A IS AN ALMOST BLOCK
C                      DIAGONAL MATRIX AND V IS A VECTOR.
C
C         PALBM      - COMPUTES B^T A, WHERE A AND B ARE ALMOST BLOCK
C                      DIAGONAL MATRICES. THE OUTPUT IS STORED IN A
C                      BAND MATRIX.
C
C         MODRIS     - HANDLES THE CHANGES IN THE RIGHT HANDSIDE VECTOR
C                      FOR GENERAL BOUNDARY CONDITIONS.
C
```

```
C
C       BONRIG    - HANDLES THE CHANGES IN THE RIGHT HANDSIDE VECTOR
C                   FOR GENERAL BOUNDARY CONDITIONS.
C
C       CALDER    - USES FINITE DIFFERENCE TO COMPUTE THE FUNCTION
C                   DERIVATIVE FOR BOUNDARY CONDITIONS.
C
C       PREMOD    - HANDLES THE CHANGES IN THE RIGHT HANDSIDE VECTOR
C                   FOR GENERAL BOUNDARY CONDITIONS.
C
C       MODRIG    - HANDLES THE CHANGES IN THE RIGHT HANDSIDE VECTOR
C                   FOR GENERAL BOUNDARY CONDITIONS.
C
C       RIGFUN    - COMPUTES THE RIGHT HAND SIDE VECTOR.
C
C       ARCE      - SOLVES THE ALMOST BLOCK DIAGONAL LINEAR SYSTEM,
C                   THIS ROUTINE CALLS ARCECO.
C
C       EIGENV    - COMPUTES THE EIGENVALUES AND EIGENVECTORS OF
C                   THE GENERALIZED EIGENVALUE PROBLEM.
C
C       PREFSD    - IS CALLED BY EIGENV. THIS ROUTINE COMPUTES THE
C                   EIGENVECTORS.
C
C       VEC22     - COMPUTES X^T A X. A IS AN ALMOST BLOCK DIAGONAL
C                   MATRIX, X IS A VECTOR.
C
C       ZTATV     - COMPUTES Z^T A^T V, WHERE Z IS A FULL MATRIX,
C                   A IS AN ALMOST BLOCK DIAGONAL MATRIX, V IS A
C                   VECTOR.
C
C       PATVEC    - COMPUTES A^T V, WHERE A IS AN ALMOST BLOCK
C                   DIAGONAL MATRIX, V IS A VECTOR.
C
C       ZV        - COMPUTES Z V, WHERE Z IS A FULL MATRIX, V IS
C                   A VECTOR.
C
C       ARCECO    - ALG. 603, ACM TOMS, VOL.9 (1983), PP. 376-380.
C
C       BSPDER    - ALG. 569, ACM TOMS, VOL.7 (1981), PP. 223-229.
C
C       DSCAL, DCOPY - ALG. 539, ACM TOMS, VOL.5(1979), PP.324-325.
C
C       DGEMV, DSBMV - LINPACK ROUTINES. LINPACK USERS' GUIDE, SIAM
C                      PUB., PHILADELPHIA, 1979.
C
C       IMTQL1, IMTQL2 - EISPACK ROUTINE. LECTURE NOTES IN COMPUTER
C                        SCIENCE 51, SPRING-VERLAG, NEW YORK, 1977.
C
C       F02FHF, F02SDF - NAG ROUTINES. NAG FORTRAN LIBRARY MANUAL,
C                        MARK 15, VOL. 5, NAG LTD., 1991.
C-----------------------------------------------------------------------
```

4.2 Program PARCOL

PARCOL is designed to compute the the coefficient $u_{i,j}(t)$ of the approximate solution U, (3.7), to equation (3.1) with initial condition (3.3) and Dirichlet or mixed boundary conditions.

4.2.1 The Calling Diagram

Figures 4.2 and 4.3 show the calling diagram of PARCOL. The user defined routine are shown by circles in Figure 4.2. The routines indicated by rectangles are part or the PARCOL package. The routines indicated dashed rectangles are routines from ARCECO, DASSL, EISPACK, LINPACK, NAG, etc. The parabolic problem is completely defined if we specify the right side function f, the coefficient functions of operators L_1 and L_2 in (3.1), the initial condition, and the boundary conditions. The right hand side function f must be provided through the FUNRIG routine, the coefficient functions for the operator L_1, $a_1(x,t)$, and $c_1(x,t)$, are provided through the PAX and PCX functions, and the coefficient functions for the operator L_2, $a_2(y,t)$, $b_2(y,t)$, and $c_2(y,t)$, are provided through the PAY, PBY, and PCY functions. The initial condition is given by FUNINI routine. The user must also provide the derivatives with respect to time of the boundary condition functions, i.e. $\frac{dh_0}{dt}(x,t)$, $\frac{dg_0}{dt}(y,t)$, $\frac{dh_1}{dt}(x,t)$, $\frac{dg_1}{dt}(y,t)$. These routines are named TPH0, TPH1, TPG0 and TPG1. To compute the coefficients $u_{0,0}(t)$, $u_{0,M_2+1}(t)$, $u_{M_1+1,0}(t)$ and $u_{M_1+1,M_2+1}(t)$, the values $g_{0,y}(0,t)$, $h_{1,x}(0,t)$, $g_{1,y}(0,t)$ and $h_{0,x}(1,t)$ are needed. These values can either be supplied by the user or approximated by finite differences. We should realize that the lower order of accuracy of these finite difference approximations will effect the accuracy of the solution. The routines PG0DY, PH1DX, PG1DY and PH0DX define $g_{0,y}(0,t)$, $h_{1,x}(0,t)$, $g_{1,y}(0,t)$ and $h_{0,x}(1,t)$. These routines are not shown in Figure 4.2 for simplicity. Similarly, in order to compute $u'_{0,0}(t)$, $u'_{0,M_2+1}(t)$, $u'_{M_1+1,0}(t)$ and $u'_{M_1+1,M_2+1}(t)$, the values $g_{0,y,t}(0,t)$, $h_{1,x,t}(0,t)$, $g_{1,y,t}(0,t)$ and $h_{0,x,t}(1,t)$ are needed.

The routines PH0DTX, PH1DTX, PG0DTY and PG1DTY serve to define these values. The BLAS routines are used in various places through PARCOL but are not shown in Figure 4.2 in order to simplify the diagram.

A main program is needed in order to call PARCOL. In this main program we need to allocate real and integer work storage arrays. The real work storage required includes two real arrays, RWORK and RPAR. The dimension of RWORK must be no less than

$$42 + 11k_1N_1k_2N_2 - 2k_1N_1k_2 + k_1N_1k_2^2N_2.$$

The dimension of RPAR must be no less than

$$84 + 31k_1N_1 + 17k_2N_2 + k_1^2N_1^2 + k_1(81 + 17k_1)/2 + N_1 + N_2 + 2k_1N_1k_2N_2 + 11k_1^2N_1 + k_2(k_2 + 7)/2 + 5k_2^2N_2.$$

The integer work storage required also includes two arrays, IWORK and IPAR. The dimension of IWORK must be no less than $20 + k_1N_1k_2N_2$, and the dimension of IPAR must be no less than $100 + 3N_1 + 3N_2 + 2k_1N_1 + 2k_2N_2$. Also PARCOL needs two arrays, Y and YPRIME, of length $(K_1N_1+2)(k_2N_2+2)$, respectively to store the coefficients and the derivatives of the coefficient with respect to t, where k_1 and k_2 are numbers of collocation points in each subinterval and N_1 and N_2 are the numbers of subintervals along x and y directions respectively. Program EVALSN is called in order to evaluate the solution at a given time T after the coefficients, $u_{ij}(T)$, $i = 1, \cdots, M_1$, $j = 1, \cdots, M_2$, are obtained. Similar to ELLDCM, the values defined by (2.82)-(2.88) are returned by PARCOL. These values are used to define the basis functions.

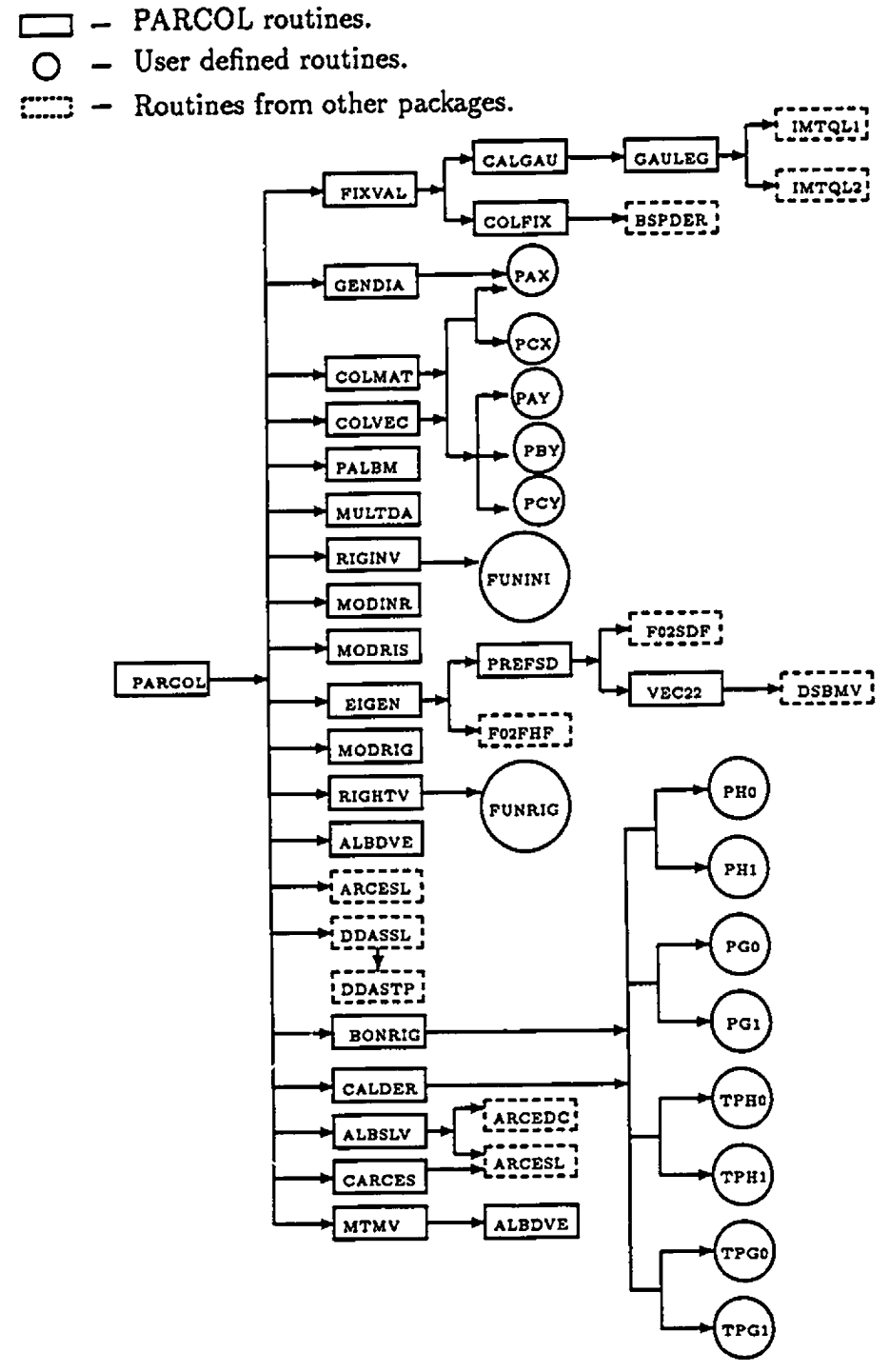

Figure 4.2: The calling tree of PARCOL.

▭ – PARCOL routines.
▭ – Routines from other packages.

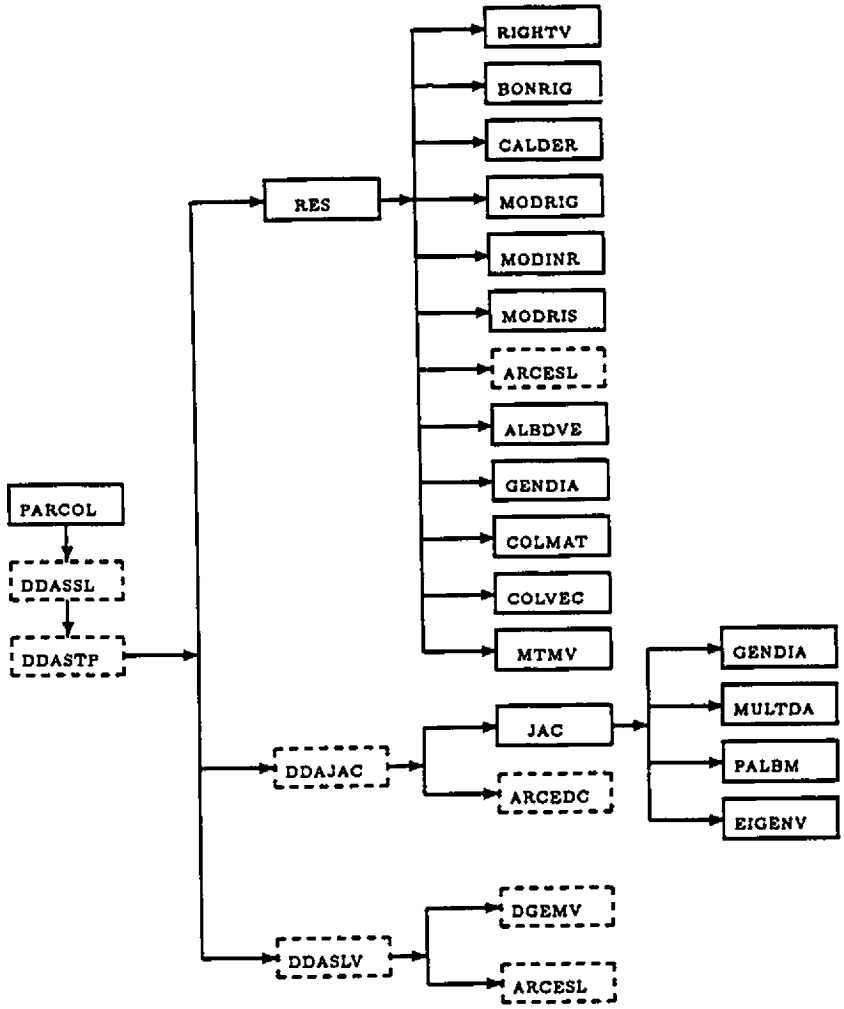

Figure 4.3: The calling tree of PARCOL, DASSL part.

4.2.2 Using PARCOL

In this section we give the comment section in PARCOL.

```
      SUBROUTINE PARCOL(KX, NBLOCX, KY, NBLOCY, TDEPCX, TDEPCY,
     1   TOUT, PAX, PCX, PAY, PBY, PCY, FUNINT, FUNRIG, AO, BO,
     2   A1, B1, RO, DO, R1, D1, PHO, PH1, PGO, PG1, BFLAG, PHODX,
     3   PH1DX, PGODY, PG1DY, TBFLAG, PHODT, PH1DT, PGODT, PG1DT,
     4   PHODTX, PH1DTX, PGODTY, PG1DTY, RTOL, ATOL, Y, YPRIME,
     5   NY, CONTX, CONTY, RWORK, NRWORK, RPAR, NRPAR, IWORK,
     6   NIWORK, IPAR, NIPAR)
C
C
C***********************************************************************
C
C     PURPOSE
C
C     THIS PROGRAM SOLVES PARABOLIC PARTIAL DIFFERENTIAL EQUATIONS
C     OF THE FORM
C
C          DU/DT = (  L  + L  ) U + F(X,Y,T),
C                     1    2
C
C     THE INITIAL CONDITION IS
C
C             U(X,Y,0) = G(X,Y),
C
C     THE BOUNDARY CONDITIONS ARE
C
C             AO U(0,Y,T) - BO U_X (0,Y,T) = PGO(Y,T),
C             A1 U(1,Y,T) - B1 U_X (1,Y,T) = PG1(Y,T),
C             RO U(X,0,T) - DO U_Y (X,0,T) = PHO(X,T),
C             R1 U(X,1,T) - D1 U_Y (X,1,T) = PH1(X,T),
C
C     THE DOMAIN OF THE EQUATION IS OMEGA = [0,1]*[0,1]
C
C                        2                              2
C     WHERE L = AX(X,T)D  + CX(X,T), L = AY(Y,T)D  + BY(Y,T)D  + CY(Y,T).
C           1          X             2          Y          Y
C
C     FOR ALL T, AX(X,T)>0 FOR X IN [0,1] AND AY(Y,T)>0 FOR Y IN [0,1]
C
C     THE PROGRAM ALSO RETURNS THE SCALAR VALUES IN EQUATIONS
C     (2.82)-(2.89) WHICH ARE USED FOR DEFINING THE BASIS FUNCTIONS.
C
C***********************************************************************
C
C                  ******** INPUT TO PARCOL ********
C
C     *** ON ENTRY ***
C
```

```
C         KX        - INTEGER
C                     THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                     IN THE X DIRECTION.
C
C         NBLOCX    - INTEGER
C                     THE NUMBER OF SUBINTERVALS IN THE X DIRECTION.
C
C         KY        - INTEGER
C                     THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                     IN THE Y DIRECTION.
C
C         NBLOCX    - INTEGER
C                     THE NUMBER OF SUBINTERVALS IN THE Y DIRECTION.
C
C         TDEPCX    - INTEGER
C                     THE NUMBER TO INFORM PARCOL IF WE HAVE TIME
C                     DEPENDENT COEFFICIENTS IN THE X DIRECTION.
C                     SET TDEPCX=0 IF THE COEFFICIENTS ARE TIME
C                     DEPENDENT, OTHERWISE SET TDEPCX TO BE NONZERO.
C
C         TDEPCY    - INTEGER
C                     THE NUMBER TO INFORM PARCOL WHETHER WE HAVE TIME
C                     DEPENDENT COEFFICIENTS IN THE Y DIRECTION.
C                     SET TDEPCY=0 IF THE COEFFICIENTS ARE TIME
C                     DEPENDENT, OTHERWISE SET TDEPCY TO BE NONZERO.
C
C         TOUT      - DOUBLE PRECISION VARIABLE
C                     THE TIME AT WHICH A SOLUTION IS DESIRED.
C
C         AO, BO    - DOUBLE PRECISION VARIABLES
C                     THE CONSTANTS TO DEFINE THE LEFT SIDE BOUNDARY
C                     CONDITIONS.
C
C         A1, B1    - DOUBLE PRECISION VARIABLES
C                     THE CONSTANTS TO DEFINE THE RIGHT SIDE BOUNDARY
C                     CONDITIONS.
C
C         RO, DO    - DOUBLE PRECISION VARIABLES
C                     THE CONSTANTS TO DEFINE THE BOTTOM SIDE BOUNDARY
C                     CONDITIONS.
C
C         R1, D1    - DOUBLE PRECISION VARIABLES
C                     THE CONSTANTS TO DEFINE THE TOP SIDE BOUNDARY
C                     CONDITIONS.
C
C         BFLAG     - INTEGER
C                     IF THE USER CAN PROVIDE THE ROUTINES TO EVALUATE
C                     D(PGO(Y,T))/DY, D(PG1(Y,T))/DY, D(PHO(X,T))/DX
C                     AND D(PH1(X,T))/DX, SET BFLAG = 1. OTHERWISE
C                     SET BFLAG = 0 AND THESE VALUES WILL THEN BE
C                     APPROXIMATED BY THE FINITE DIFFERENCES.
C
C         TBFLAG    - INTEGER
```

```
C                     IF THE USER CAN PROVIDE THE ROUTINES TO EVALUATE
C                     D(PGODT(Y,T))/DY, D(PG1DT(Y,T))/DY,
C                     D(PHODT(X,T))/DX AND D(PH1DT(X,T))/DX, SET
C                     TBFLAG = 1. OTHERWISE SET TBFLAG = 0, THESE
C                     VALUES WILL THEN BE APPROXIMATED BY THE FINITE
C                     DIFFERENCES.
C
C         ATOL      - DOUBLE PRECISION VARIABLE
C                     THE ABSOLUTE ERROR TOLERANCE FOR THE ODE SOLVER.
C                     SEE DASSL DOCUMENTATION FOR FURTHER EXPLANATION.
C
C         RTOL      - DOUBLE PRECISION VARIABLE
C                     THE RELATIVE ERROR TOLERANCE FOR THE ODE SOLVER.
C                     SEE DASSL DOCUMENTATION FOR FURTHER EXPLANATION.
C
C         RWORK     - DOUBLE PRECISION ARRAY OF LENGTH NRWORK.
C                     WORK STORAGE.
C
C         NRWORK    - INTEGER
C                     THE LENGTH OF RWORK
C                     NWORK >= 42 + 11 KX NBLOCX KY NBLOCY -
C                           2 KX NBLOCX KY + KX NBLOCX KY**2 NBLOCY
C
C         RPAR      - DOUBLE PRECISION ARRAY OF LENGTH NRPAR.
C                     WORK STORAGE.
C
C         NRPAR     - THE LENGTH OF RPAR
C                     NRPAR >= 84 + 31 KX NBLOCX + 17 KY NBLOCY +
C                           KX**2 NBLOCX**2 + ((81+17*KXTOP) KXTOP)/2+
C                           NBLTOX+NBLTOY+2 KXTOP NBLTOX KYTOP NBLTOY+
C                           11 KXTOP**2 NBLTOX + ((7+KYTOP) KYTOP)/2 +
C                           5 KYTOP**2 NBLTOY
C
C         IPAR      - INTEGER ARRAY OF LENGTH NIPAR.
C                     INTEGER WORK STORAGE.
C
C         NIPAR     - INTEGER
C                     THE LENGTH OF IPAR
C                     NIPAR >= 100 + 3 NBLOCY + 3 KX NBLOCX +
C                              2 KXTOP NBLTOX + 2 KY NBLOCY.
C
C         IWORK     - INTEGER ARRAY OF LENGTH NIWORK.
C                     INTEGER WORK STORAGE.
C
C         NIWORK    - INTEGER
C                     THE LENGTH OF IWORK
C                     NIWORK >= 20 + KX NBLOCX KY NBLOCY.
C
C*********************************************************************
C
C                   ******** OUTPUT FROM ELLDCM ********
C
C         Y          - DOUBLE PRECISION ARRAY OF LENGTH NY.
```

```
C                    THE COEFFICIENTS OF THE APPROXIMATE SOLUTION.
C
C         YPRIME     - DOUBLE PRECISION ARRAY OF LENGTH NY.
C                      THE DERIVATIVE OF THE COEFFICIENTS WITH
C                      RESPECT TO T.
C
C         NY         - INTEGER
C                      THE LENGTH OF Y AND YPRIME
C                      NY = (KX*NBLOCX+2)*(KY*NBLOCY+2)
C
C         CONTX      - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                      THE SCALAR CONSTANTS GIVEN BY (2.82)-(2.85)
C                      FOR DEFINING THE BASIS FUNCTIONS.
C
C         CONTY      - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                      THE SCALAR CONSTANTS GIVEN BY (2.86)-(2.89)
C                      FOR DEFINING THE BASIS FUNCTIONS.
C
C*************************************************************************
C
C                    ******** USER DEFINED FUNCTIONS ***********
C
C       THE FOLLOWING USER DEFINED FUNCTIONS SHOULD BE DECLARED AS
C       EXTERNAL IN THE MAIN PROGRAM. THE ARGUMENTS FOR EACH FUNCTION
C       ARE DOUBLE PRECISION VARIABLES X, Y AND T.
C
C          PAX        - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                       FUNCTION AX IN OPERATOR LX. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PAX(X, T)
C
C          PCX        - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                       FUNCTION CX IN OPERATOR LX. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PCX(X, T)
C
C          PAY        - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                       FUNCTION AY IN OPERATOR LY. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PAY(X, T)
C
C          PBY        - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                       FUNCTION BY IN OPERATOR LY. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PBY(X, T)
C
C          PCY        - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                       FUNCTION CY IN OPERATOR LY. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PCY(X, T)
C
C          PGO        - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                       BOUNDARY CONDITIONS AT THE THE RIGHT HAND
C                       OF THE REGION. THE HEADING IS
C                       DOUBLE PRECISION FUNCTION PGO(X, T)
C
C          PG1        - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                       BOUNDARY CONDITIONS AT THE THE LEFT HAND
```

```
C                 OF THE REGION. THE HEADING IS
C                 DOUBLE PRECISION FUNCTION PG1(X, T)
C
C       PHO       - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                   BOUNDARY CONDITIONS AT THE THE BOTTOM SIDE
C                   OF THE REGION. THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PHO(X, T)
C
C       PH1       - DOUBLE PRECISION FUNCTION TO DEFINE THE
C                   BOUNDARY CONDITIONS AT THE THE TOP SIDE
C                   OF THE REGION. THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PH1(X, T)
C
C       PHODX     - DOUBLE PRECISION FUNCTION, PHODX = D(PHO)/DX.
C                   WHEN BFLAG = 0, IGNORE THIS PARAMETER BY
C                   TREATING IT AS A DUMMY ARGUMENT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PHODX(X, T)
C
C       PH1DX     - DOUBLE PRECISION FUNCTION, PH1DX = D(PH1)/DX.
C                   WHEN BFLAG = 0, IGNORE THIS PARAMETER BY
C                   TREATING IT AS A DUMMY ARGUMENT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PH1DX(X, T)
C
C       PGODY     - DOUBLE PRECISION FUNCTION, PGODY = D(PGO)/DY.
C                   WHEN BFLAG = 0, IGNORE THIS PARAMETER BY
C                   TREATING IT AS A DUMMY ARGUMENT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PGODY(X, T)
C
C       PG1DY     - DOUBLE PRECISION FUNCTION, PG1DY = D(PG1)/DY.
C                   WHEN BFLAG = 0, IGNORE THIS PARAMETER BY
C                   TREATING IT AS A DUMMY ARGUMENT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PG1DY(X, T)
C
C       PHODT     - DOUBLE PRECISION FUNCTION, PHODT = D(PHO)/DT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PHODT(X, T)
C
C       PH1DT     - DOUBLE PRECISION FUNCTION, PH1DT = D(PH1)/DT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PH1DT(X, T)
C
C       PGODT     - DOUBLE PRECISION FUNCTION, PGODT = D(PGO)/DT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PGODT(X, T)
C
C       PG1DT     - DOUBLE PRECISION FUNCTION, PG1DT = D(PG1)/DT.
C                   THE HEADING IS
C                   DOUBLE PRECISION FUNCTION PG1DT(X, T)
C
```

```
C         PHODTX    - DOUBLE PRECISION FUNCTION, PHODTX = D(PHODT)/DX.
C                     WHEN TBFLAG = 0, IGNORE THIS PARAMETER BY
C                     TREATING IT AS A DUMMY ARGUMENT.
C                     THE HEADING IS
C                     DOUBLE PRECISION FUNCTION PHODTX(X, T)
C
C         PH1DTX    - DOUBLE PRECISION FUNCTION, PH1DTX = D(PH1DT)/DX.
C                     WHEN TBFLAG = 0, IGNORE THIS PARAMETER BY
C                     TREATING IT AS A DUMMY ARGUMENT.
C                     THE HEADING IS
C                     DOUBLE PRECISION FUNCTION PH1DTX(X, T)
C
C         PGODTY    - DOUBLE PRECISION FUNCTION, PGODTY = D(PGODT)/DY.
C                     WHEN TBFLAG = 0, IGNORE THIS PARAMETER BY
C                     TREATING IT AS A DUMMY ARGUMENT.
C                     THE HEADING IS
C                     DOUBLE PRECISION FUNCTION PGODTY(X, T)
C
C         PG1DTY    - DOUBLE PRECISION FUNCTION, PG1DTY = D(PG1DT)/DY.
C                     WHEN TBFLAG = 0, IGNORE THIS PARAMETER BY
C                     TREATING IT AS A DUMMY ARGUMENT.
C                     THE HEADING IS
C                     DOUBLE PRECISION FUNCTION PG1DTY(X, T)
C
C         FUNINI    - DOUBLE PRECISION FUNCTION TO EVALUATE THE
C                     INITIAL FUNCTION G OF THE DIFFERENTIAL
C                     EQUATION. THE HEADING IS
C                     DOUBLE PRECISION FUNCTION FUNINI(X, Y)
C
C         FUNRIG    - DOUBLE PRECISION FUNCTION TO EVALUATE THE RIGHT
C                     HAND SIDE FUNCTION F OF THE DIFFERENTIAL
C                     EQUATION. THE HEADING IS
C                     DOUBLE PRECISION FUNCTION FUNRIG(X, Y, T)
C
C*********************************************************************
C
C                  ******** PACKAGE SUBROUTINES ***********
C
C         PARCOL    - CHECKS STORAGE PARAMETERS, BREAKS UP THE WORK
C                     AREA. THIS ROUTINE IS THE DRIVER FOR THE
C                     PACKAGE. IT CALLS FIXVAL, COLMAT, COLVEC, DCOPY,
C                     GENDIA, MULTDA, PALBM, EIGENV, BONRIG, RIGINV,
C                     CALDER, MODINR, ALBSLV, MODRIS, ARCESL, ALBDVE,
C                     RIGHTV, MODRIG, MTMV, CARCES AND DDASSL.
C
C         FIXVAL    - CALCULATES THE FIXED VALUES WHICH ARE NEEDED
C                     BY PARCOL. THESE VALUES DEFINE THE MESH POINTS,
C                     THE COLLOCATION POINTS, THE COLLOCATION MATRIX
C                     AND THE FIRST AND SECOND DERIVATIVE OF THE
C                     COLLOCATION MATRIX.
C
C         COLFIX    - IS CALLED BY FIXVAL. THIS ROUTINE CALCULATE
C                     THE COLLOCATION MATRIX, FIRST AND SECOND
```

```
C                    DERIVATIVE OF THE COLLOCATION MATRIX.
C
C          GENDIA    - COMPUTES THE DIAGONAL MATRIX W*D DEFINED IN
C                      LEMMA 2.1.
C
C          COLMAT    - COMPUTES THE COLLOCATION MATRIX AX AND AY.
C
C          COLVEC    - COMPUTES THE TIME DEPENDENT VALUES WHICH ARE
C                      USED TO HANDLE BOUNDARY CONDITIONS.
C
C          RIGINV    - COMPUTES THE INITIAL VALUE VECTORS FOR GETTING
C                      THE INITIAL VALUES OF ODE.
C
C          MODINR    - HANDLES THE BOUNDARY CONDITIONS.
C
C          RIGHTV    - COMPUTES THE RIGHT HAND SIDE VECTOR.
C
C          DDASSL    - THE DRIVER ROUTINE FOR DASSL, THE DAE SOLVER.
C
C          DDAJAC    - THE ROUTINE FROM DASSL TO COMPUTE THE ITERATION
C                      MATRIX AND FORM THE 'LU'-DECOMPOSITION.
C
C          DDASLV    - THE ROUTINE FROM DASSL TO THAT MANAGES THE
C                      SOLUTION OF THE LINEAR SYSTEM  ARISING IN THE
C                      NEWTON ITERATION.
C
C          MTMV      - COMPUTES (A TENSOR B)*V, WHERE A AND B ARE
C                      ALMOST BLOCK DIAGONAL MATRICES, V IS A VECTOR.
C
C          DRES      - DEFINES THE ODES.
C
C          DJAC      - RETURNS THE JACOBIAN MATRICES OF THE ODE.
C
C          ALBSLV    - SOLVES THE LINEAR SYSTEM IN THE FORM OF
C                      (A TENSOR B)*X = C. THIS ROUTINE CALLS
C                      ARCEDC AND ARCESL.
C
C          CARCESL   - SOLVES THE LINEAR SYSTEM IN THE FORM OF
C                      (A TENSOR B)*X = C, WHERE A AND B ARE ALREADY
C                      FACTORIZED BY ARCEDC. THIS ROUTINE CALLS ARCESL
C                      ONLY.
C
C          ARCEDC    - ARCECO'S FACTORIZATION ROUTINE.
C
C          ARCESL    - ARCECO'S BACK SUBSTITUTION ROUTINE.
C
C          THE DESCRIPTIONS OF CALGAU, MULTDA, PALBM, MODRIS, EIGENV,
C          PREFSD, VEC22, MODRIG, ALBDVE, BONRIG, CALDER, BSPDER,
C          DSCAL, DCOPY, DGEMV, DSBMV, IMTQL1, IMTQL2, FO2FHF AND
C          FO2SDF CAN BE FOUND IN THE INITIAL COMMENTS OF ELLDCM.
C-----------------------------------------------------------------
```

4.3 Program EVALSN

The program EVALSL takes the output $\{u_{i,j}\}$ from ELLDCM or PARCOL and evaluates the approximate solution at arbitrary (x_1, x_2). Note that EVALSN needs the values, (2.82)-(2.88), in order to define the basis functions. These values are returned by ELLDCM or PARCOL. Here we give the comment section from EVALSL.

```
      SUBROUTINE EVALSN(XARRAY, SARRAY, DXARAY, DYARAY, YP, COEFF,
     *   XI, YI, NP, NBLOCX, KX, NBLOCY, KY, CONTX, CONTY, WORKX,
     *   WORKY, WORKXY, IEVAL)

C
C***********************************************************************
C
C     PURPOSE
C
C     EVALUATE THE SOLUTION.
C
C***********************************************************************
C
C                  ******** INPUT TO EVALSL ********
C
C         XARRAY    - DOUBLE PRECISION ARRAY OF LENGTH NP.
C                     THE X VALUES OF THE POINTS IN THE X DIRECTION
C                     AT WHICH THE SOLUTION IS DESIRED.
C
C         YP        - DOUBLE PRECISION VARIABLE.
C                     THE Y VALUE OF THE POINTS AT WHICH THE SOLUTION
C                     IS DESIRED.
C
C         COEFF     - DOUBLE PRECISION ARRAY OF LENGTH (KX*NBLOCX+2)*
C                     (KY*NBLOCY+2).
C                     THE COEFFICIENT ARRAY RETURNED FROM ELLDCM OR
C                     PARCOL.
C
C         XI        - DOUBLE PRECISION ARRAY OF LENGTH NBLOCX+1.
C                     THE X MESH.
C
C         YI        - DOUBLE PRECISION ARRAY OF LENGTH NBLOCY+1.
C                     THE Y MESH.
C
C         NP        - INTEGER
C                     THE NUMBER OF POINTS AT WHICH THE SOLUTION IS
C                     DESIRED.
C
C         KX        - INTEGER
C                     THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                     IN THE X DIRECTION.
C
```

```
C         NBLOCX    - INTEGER
C                     THE NUMBER OF SUBINTERVALS IN THE X DIRECTION.
C
C         KY        - INTEGER
C                     THE NUMBER OF COLLOCATION POINTS PER SUBINTERVAL
C                     IN THE Y DIRECTION.
C
C         NBLOCY    - INTEGER
C                     THE NUMBER OF SUBINTERVALS IN THE Y DIRECTION.
C
C         CONTX     - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                     THE SCALAR CONSTANTS GIVEN BY (2.82)-(2.85),
C                     FOR DEFINING THE BASIS FUNCTIONS. THESE VALUES
C                     ARE RETURNED FROM ELLDCM OR PARCOL.
C
C         CONTY     - DOUBLE PRECISION ARRAY OF LENGTH 8.
C                     THE SCALAR CONSTANTS GIVEN BY (2.86)-(2.89),
C                     FOR DEFINING THE BASIS FUNCTIONS. THESE VALUES
C                     ARE RETURNED FROM ELLDCM OR PARCOL.
C
C         WORKX     - DOUBLE PRECISION ARRAY
C                     REAL WORK STORAGE. THE LENGTH OF WORKX MUST
C                     BE NO LESS THAN ((3+KX)*(2+KX))/2+3*(2+KX).
C
C         WORKY     - DOUBLE PRECISION ARRAY
C                     REAL WORK STORAGE. THE LENGTH OF WORKY MUST
C                     BE NO LESS THAN ((3+KY)*(2+KY))/2+3*(2+KY).
C
C         WORKXY    - DOUBLE PRECISION ARRAY
C                     REAL WORK STORAGE. THE LENGTH OF WORK MUST
C                     BE NO LESS THAN KX*NBLOCX.
C
C         IEVAL     - AN INTEGER INDICATING WHETHER THE SOLUTION AND
C                     ITS DERIVATIVE SHOULD BE EVALUATED
C                     IEVAL = 0, FOR FUNCTION VALUES ONLY
C                     IEVAL = 1, FOR FUNCTION VALUES AND DERIVATIVES.
C
C*************************************************************************
C
C                 ******** OUTPUT FROM ELLDCM ********
C
C         SARRAY    - DOUBLE PRECISION ARRAY OF LENGTH NP.
C                     THE APPROXIMATE SOLUTION VALUES AT (XARRAY, YP)
C
C         DXARAY    - DOUBLE PRECISION ARRAY OF LENGTH NP.
C                     THE DERIVATIVE OF THE SOLUTION WITH RESPECT
C                     TO X
C
C         DYARAY    - DOUBLE PRECISION ARRAY OF LENGTH NP.
C                     THE DERIVATIVE OF THE SOLUTION WITH RESPECT
C                     TO Y
C
C-------------------------------------------------------------------------
```

Chapter 5

Concluding Remarks and Future Work

In this thesis we have described a parallel collocation algorithm based on orthogonal spline collocation and matrix decomposition techniques for solving partial differential equations of elliptic type. We have implemented this algorithm on a parallel computer system, the Alliant/FX2800, and have demonstrated the parallel speedup of our algorithm through numerical examples. The efficiency of our software was demonstrated by a comparison with another popular package SERRG2 [41]. We have proposed and implemented a parallel collocation algorithm for solving partial differential equations of parabolic type. The algorithm is based on a method-of-lines approach in which orthogonal spline collocation and matrix decomposition techniques are incorporated. The differential/algebraic solver DASSL [59] was employed to solve the ODE systems introduced by the collocation. An ABD solver ARCECO [20] was added to DASSL to efficiently handle the special structure of the Jacobian matrix associated with the ODE system. This approach greatly improves the performance of DASSL. We have shown that our algorithm can achieve nearly linear speedups in parallel computation mode through numerical experiments on an Alliant/FX2800 system.

Future work will continue in several directions. We plan to implement our software on other parallel computer systems. Parallelism is exploitable at several levels of

granularity. Hence a more massively parallel system could be usefully employed.

Another area of future investigation includes the different choice of the spline basis. In our algorithm, B-spline basis functions [14] are used for the spatial discretization component. With B-spline basis functions, the continuity conditions are built into the basis functions. Ascher, Pruess and Russell [5] have shown, however, that monomial basis functions have certain advantages over the B-splines. For example, compared to the B-splines, monomial splines are easier and cheaper to implement. In our future research we will consider using monomial splines for the spatial discretization component. When these basis functions are used, the linear systems which arise still have a special almost block diagonal structure. For such special ABD structure, algorithm [52], similar to those employed in ARCECO, can be employed.

Another area of future work is to study the possibility of decoupling the ODEs, as mentioned in Section 3.2. Solving each of the ODE subsystems independently may lead to efficiency improvements.

Another area of future work is to extend our algorithms both for elliptic and parabolic cases for problems with higher space dimensions. It should be noted that three or more dimensions can be handled in a similar way without increasing computation complexity, [48].

Another area of future research is to extend our algorithms to handle certain non-linear PDEs. The algorithms which we have constructed are restricted to linear separable problems, with linear boundary conditions. Extending the matrix decomposition technique to linear non-separable or non-linear elliptic problems, or linear problems with non-linear boundary conditions is not trivial. If a non-linear elliptic problem is approximated by collocation, a system of non-linear equations results. The question is whether the structure of the Jacobian of the non-linear system allows the matrix decomposition technique to be used. Similarly, if collocation is used on a non-linear parabolic PDE, the problem is whether the Jacobian of the non-linear equations which are solved at each time step has the same structure as the Jacobian for linear parabolic PDEs. In one space variable, this is the case, [51]. Further work

is needed to determine if the technique can be applied to two dimensional non-linear parabolic PDEs.

The matrix decomposition method as we have presented it, cannot be applied to non-separable problems. It does not seem likely that the method could be modified for such problems. Non-separable problems can be solved using Alternating Direction Implicit (ADI) Galerkin methods and Crank-Nicolson for the time stepping. An important advantage of this method over matrix decomposition is that each time step is $O(P)$, where P is the number of unknowns, which is much less than for matrix decomposition. However whereas DASSL may use higher order BDF formulas for the time stepping, the Crank-Nicolson method is restricted to $O(\Delta t)^2$. In addition the time error is not as effectively controlled as in DASSL.

Appendix A

Example Drivers

A.1 Example Driver for Testing ELLDCM

In this section we give the main program and user defined routines for solving the problem:

$$(L_1 + L_2)u = f, \tag{A.1}$$

where

$$L_1 = -\frac{\partial^2}{\partial x_1^2} \text{ and } L_2 = -\frac{\partial^2}{\partial x_2^2}, \tag{A.2}$$

and the boundary conditions are,

$$2u(0,x_2) - u_{x_1}(0,x_2) = 4e^{x_2} + 1, \;\; u(1,x_2) - u_{x_1}(1,x_2) = 2e^{x_2},$$
$$u(x_1,0) - u_{x_2}(x_1,0) = e^{x_1}, \;\; u(x_1,1) - u_{x_2}(x_1,1) = e^{x_1}.$$

The function f is chosen such that the solution is $u = 2e^{x_1} + e^{x_2}$.

```
************************************************************************
C     PROGRAM MAIN

      DOUBLE PRECISION LX, LY, ONE, DLX, DLY, ZERO, TWO, FOUR, HUDRD
      PARAMETER (ONE=1.0D0, ZERO=0.0D0, TWO=2.0D0, FOUR=4.0D0,
     *   HUDRD=100.D0, LX=ONE, LY=ONE)

      INTEGER KXTOP, KYTOP, NBLTOX, NBLTOY, M, NSLT02, BWORKX,
```

```
*    BWORKY, NWORK, NIWORK
     PARAMETER(M=2, KXTOP=2, KYTOP=2, NBLTOX=32, NBLTOY=32)

C    THE SIZE OF WORK ARRAY ETC.
     PARAMETER(NSLT02=(KXTOP*NBLTOX+2)*(KYTOP*NBLTOY+2),
*          BWORKX=((M+KXTOP+1)*(M+KXTOP))/2+(M+1)*(M+KXTOP),
*          BWORKY=((M+KYTOP+1)*(M+KYTOP))/2+(M+1)*(M+KYTOP))

     PARAMETER(NWORK=58+30*KXTOP*NBLTOX+KXTOP**2*NBLTOX**2+
*               15*KYTOP*NBLTOY+10*KXTOP**2*NBLTOX+
*               3*KYTOP**2*NBLTOY+((81+17*KXTOP)*KXTOP)/2+
*               ((7+KYTOP)*KYTOP)/2,
*          NIWORK=3*NBLTOX+3*NBLTOY+KXTOP*NBLTOX+KYTOP*NBLTOY)

C    DECLARE THE WORK ARRAY ETC.
     DOUBLE PRECISION WORK(NWORK), COEFF(NSLT02), XI(NBLTOX+1),
*    YI(NBLTOY+1), AX, CX, AY, BY, CY, FUN, G0, G1, H0, H1,
*    A0, B0, A1, B1, R0, D0, R1, D1, CONTX(8), CONTY(8),
*    DG0DY0, DH1DX0, DH0DX1, DG1DY1
     EXTERNAL AX, CX, AY, BY, CY, FUN, G0, G1, H0, H1

     INTEGER IWORK(NIWORK), KX, NBLOCX, KY, NBLOCY, NX, NY,
*    NSOL2, NBLXP1, NBLYP1, BFLAG, IEVAL, IN1, NP, J

C    FOR EVALUATION OF THE SOLUTION.
     DOUBLE PRECISION  XPOINT(200), YPOINT(200), SARRAY(200),
*    DXARAY(200), DYARAY(200), LINFSL, LINFER, EXACTP, Y1,
*    SOLERR, DXERR, DYERR, LERRDX, LERRDY, LERDXY, TEMP,
*    EXACT, DXEXAT, DYEXAT, WORKX(BWORKX), WORKY(BWORKY),
*    WORKXY(KXTOP*NBLTOX+2), ABS, EXP
     EXTERNAL EXACT, DXEXAT, DYEXAT
     INTRINSIC ABS, EXP

C    INPUT DATA.
     READ(5,*) KX, NBLOCX
     READ(5,*) KY, NBLOCY
     READ(5,*) BFLAG
     READ(5,*) A0, B0, A1, B1
     READ(5,*) R0, D0, R1, D1

C    DEFINE THE VALUES IF THE FINITE DIFFERENCE ARE NOT USED.
     IF(BFLAG .EQ. 1) THEN
        DG0DY0 = FOUR
        DG1DY1 = TWO*EXP(ONE)
        DH0DX1 = EXP(ONE)
        DH1DX0 = ONE
     ENDIF

C    SETUP UP THE SIZE OF COLLOCATION MATRIX, MESH SIZE, AND
C    TOTAL NUMBER OF UNKNOWNS.
     NX = KX*NBLOCX
     NY = KY*NBLOCY
     NBLXP1 = NBLOCX+1
```

```
      NBLYP1 = NBLOCY+1
      NSOL2 = (NX+2)*(NY+2)

C     GET THE STEP SIZE.
      DLX = LX/FLOAT(NBLOCX)
      DLY = LY/FLOAT(NBLOCY)

C     SET UP MESH.
      DO 10 IN1=1,NBLXP1
         XI(IN1) = (FLOAT(IN1)-ONE)*DLX
   10 CONTINUE
      DO 20 IN1=1,NBLYP1
         YI(IN1) = (FLOAT(IN1)-ONE)*DLY
   20 CONTINUE

      WRITE(6,*)'KX, NBLOCX ARE ', KX, NBLOCX
      WRITE(6,*)'KY, NBLOCY ARE ', KY, NBLOCY
      WRITE(6,*)'A0, B0, A1, B1 ', A0, B0, A1, B1
      WRITE(6,*)'R0, D0, R1, D1 ', R0, D0, R1, D1

      CALL ELLDCM(KX, NBLOCX, KY, NBLOCY, AX, CX, AY, BY, CY,
     1   FUN, A0, B0, A1, B1, R0, D0, R1, D1, H0, H1, G0, G1,
     2   BFLAG, DG0DY0, DH1DX0, DH0DX1, DG1DY1, COEFF, NSOL2,
     3   XI, YI, CONTX, CONTY, WORK, NWORK, IWORK, NIWORK)

C     SETUP FOR COMPUTATION OF THE MAXIMUM ERROR AT MESH POINTS.
      LINFER = ZERO
      LINFSL = ZERO
      LERRDX = ZERO
      LERRDY = ZERO
      LERDXY = ZERO

C     SETUP THE X VALUES AT WHICH THE SOLUTIONS ARE DESIRED.
      NP = NBLXP1
      DO 150 IN1=1,NBLXP1
         XPOINT(IN1) = XI(IN1)
  150 CONTINUE

C     EVALUATE THE SOLUTION AND FIND THE MAXIMUM ERROR AT MESH
C     POINTS.
      IEVAL = 1
      DO 170 J=1,NBLYP1
         Y1 = YI(J)
         CALL EVALSN(XPOINT, SARRAY, DXARAY, DYARAY,
     *      Y1, COEFF , XI, YI, NP, NBLOCX, KX, NBLOCY, KY,
     *      CONTX, CONTY, WORKX, WORKY, WORKXY, IEVAL)
C        FIND THE MAXIMUM VALUES OF ERROR.
         DO 160 IN1=1,NBLXP1
            EXACTP = EXACT(XPOINT(IN1),Y1)
            IF ( LINFER .LT. ABS(EXACTP - SARRAY(IN1)) ) THEN
               LINFER = ABS(EXACTP - SARRAY(IN1))
            ENDIF
  160    CONTINUE
```

```
        SOLERR = LINFER

C     FIND THE MAXIMUM ERROR AT MESH POINTS FOR THE DERIVATIVES.
      IF(IEVAL .NE. 0) THEN
          DO 161 IN1=1,NBLXP1
            EXACTP = DXEXAT(XPOINT(IN1),Y1)
            TEMP = ABS(EXACTP - DXARAY(IN1))
            IF ( LERRDX .LT. TEMP ) THEN
               LERRDX = TEMP
            ENDIF
  161     CONTINUE
          DXERR = LERRDX
          DO 162 IN1=1,NBLXP1
            EXACTP = DYEXAT(XPOINT(IN1),Y1)
            TEMP = ABS(EXACTP - DYARAY(IN1))
            IF ( LERRDY .LT. TEMP ) THEN
               LERRDY = TEMP
            ENDIF
  162     CONTINUE
          DYERR = LERRDY
      ENDIF
  170 CONTINUE

C     OUTPUT THE ERROR.
      PRINT *,'FUN NODE ERROR = ',SOLERR
      IF(IEVAL .GE. 1) THEN
         PRINT *,'DX  NODE ERROR = ',DXERR
         PRINT *,'DY  NODE ERROR = ',DYERR
      ENDIF

C     SETUP FOR COMPUTATION OF THE L-INFINITY ERROR.
      LINFER = ZERO
      LINFSL = ZERO
      LERRDX = ZERO
      LERRDY = ZERO
      LERRXY = ZERO

C     SETUP THE SAMPLE POINTS FOR THE L-INFINITY ERROR.
      DLX = LX/HUDRD
      DLY = LY/HUDRD
      DO 250 IN1=1,101
         XPOINT(IN1) = FLOAT((IN1-1)*DLX)
         YPOINT(IN1) = FLOAT((IN1-1)*DLY)
  250 CONTINUE

C     EVALUATE THE SOLUTION AND FIND THE L-INFINITY ERROR.
      NP = 101
      DO 180 J=1,101
         Y1 = YPOINT(J)
         CALL EVALSN(XPOINT, SARRAY, DXARAY, DYARAY,
     *      Y1, COEFF, XI, YI, NP, NBLOCX, KX, NBLOCY, KY,
     *      CONTX, CONTY, WORKX, WORKY, WORKXY, IEVAL)
         DO 190 IN1=1,101
```

```
            EXACTP = EXACT(XPOINT(IN1),Y1)
            TEMP = ABS(EXACTP - SARRAY(IN1))
            IF ( LINFER .LT. TEMP ) THEN
               LINFER = TEMP
            ENDIF
  190     CONTINUE
          SOLERR = LINFER

C       FIND THE L-INFINITY ERROR FOR THE DERIVATIVES.
        IF(IEVAL .NE. 0) THEN
           DO 181 IN1=1,101
              EXACTP = DXEXAT(XPOINT(IN1),Y1)
              TEMP = ABS(EXACTP - DXARAY(IN1))
              IF ( LERRDX .LT. TEMP ) THEN
                 LERRDX = TEMP
              ENDIF
  181     CONTINUE
          DXERR = LERRDX
           DO 182 IN1=1,101
             EXACTP = DYEXAT(XPOINT(IN1),Y1)
             TEMP = ABS(EXACTP - DYARAY(IN1))
             IF ( LERRDY .LT. TEMP ) THEN
                LERRDY = TEMP
             ENDIF
  182     CONTINUE
          DYERR = LERRDY
        ENDIF
  180 CONTINUE
        PRINT *,'FUN MAX  ERROR = ',SOLERR
        IF(IEVAL .GE. 1) THEN
           PRINT *,'DX  MAX  ERROR = ',DXERR
           PRINT *,'DY  MAX  ERROR = ',DYERR
        ENDIF
        PRINT *

        STOP
        END
```

The following functions define the operators L_1 and L_2.

```
        DOUBLE PRECISION FUNCTION AX(X)
        DOUBLE PRECISION X, ONE
        PARAMETER ( ONE = 1.0D0 )
        AX = ONE
        RETURN
        END

        DOUBLE PRECISION FUNCTION CX(X)
        DOUBLE PRECISION X, ZERO
        PARAMETER ( ZERO = 0.0D0 )
        CX = ZERO
        RETURN
        END
```

```
      DOUBLE PRECISION FUNCTION AY(X)
      DOUBLE PRECISION X, ONE
      PARAMETER ( ONE = 1.0D0 )
      AY = ONE
      RETURN
      END

      DOUBLE PRECISION FUNCTION BY(X)
      DOUBLE PRECISION X, ZERO
      PARAMETER ( ZERO = 0.0D0 )
      BY = ZERO
      RETURN
      END

      DOUBLE PRECISION FUNCTION CY(X)
      DOUBLE PRECISION X, ZERO
      PARAMETER ( ZERO = 0.0D0 )
      CY = ZERO
      RETURN
      END
```

The following function gives the right hand side function of the differential equation.

```
      DOUBLE PRECISION FUNCTION FUN(X,Y)
      DOUBLE PRECISION X, Y, TWO, EXP
      INTRINSIC EXP
      PARAMETER ( TWO=2.0D0 )
      FUN = -(EXP(X)+TWO*EXP(Y))
      RETURN
      END
```

The following functions give the boundary conditions.

```
      DOUBLE PRECISION FUNCTION GO(X)
      DOUBLE PRECISION X, ONE, FOUR, EXP
      INTRINSIC EXP
      PARAMETER ( ONE = 1.0D0, FOUR=4.0D0 )
      GO = FOUR*EXP(X)+ONE
      RETURN
      END

      DOUBLE PRECISION FUNCTION G1(X)
      DOUBLE PRECISION X, TWO, EXP
      INTRINSIC EXP
      PARAMETER ( TWO=2.0D0 )
      G1= TWO*EXP(X)
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION H0(X)
      DOUBLE PRECISION X, EXP
      INTRINSIC EXP
      H0 = EXP(X)
      RETURN
      END

      DOUBLE PRECISION FUNCTION H1(X)
      DOUBLE PRECISION X, EXP
      INTRINSIC EXP
      H1= EXP(X)
      RETURN
      END
```

The following functions give the solution and the derivatives of the solution.

```
      DOUBLE PRECISION FUNCTION EXACT(X,Y)
      DOUBLE PRECISION X, Y, TWO, EXP
      INTRINSIC EXP
      PARAMETER ( TWO=2.0D0 )
      EXACT=EXP(X)+TWO*EXP(Y)
      RETURN
      END

      DOUBLE PRECISION FUNCTION DXEXAT(X,Y)
      DOUBLE PRECISION X, Y, EXP
      INTRINSIC EXP
      DXEXAT=EXP(X)
      RETURN
      END

      DOUBLE PRECISION FUNCTION DYEXAT(X,Y)
      DOUBLE PRECISION X, Y, TWO, EXP
      INTRINSIC EXP
      PARAMETER ( TWO=2.0D0 )
      DYEXAT=TWO*EXP(Y)
      RETURN
      END
```

Some sample input data:

```
2 32
2 32
1
2.D0 1.D0 1.D0 1.D0
1.D0 1.D0 1.D0 1.D0
```

The corresponding output:

```
KX, NBLOCX ARE   2  32
KY, NBLOCY ARE   2  32
A0, B0, A1, B1     2.00    1.00    1.00    1.00
R0, D0, R1, D1     1.00    1.00    1.00    1.00
FUN NODE ERROR =     3.6218441579194D-09
DX  NODE ERROR =     9.8302548323659D-09
DY  NODE ERROR =     6.2531260169862D-09
FUN MAX  ERROR =     2.0926357535700D-08
DX  MAX  ERROR =     6.0558739356509D-07
DY  MAX  ERROR =     1.2198256715834D-06
```

A.2 Example Driver for Testing PARCOL

In this section we give the main program and user defined routines for solving the problem:

$$\frac{\partial u}{\partial t} = (L_1 + L_2)u + f(x_1, x_2, t), \tag{A.3}$$

where

$$\begin{aligned} L_1 &= (t+1)(x_1^2+2)\frac{\partial^2}{\partial x_1^2} + t\,\sin(\pi x_1), \\ L_2 &= (t+2)(x_2^4+5)\frac{\partial^2}{\partial x_2^2} + (\sin(\pi x_2)+t)\frac{\partial}{\partial x_2} + \cos(\pi x_2)t^2, \end{aligned}$$

the initial condition is,

$$u(x_1, x_2, 0) = 2(e^{x_1} + 2e^{x_2}), \tag{A.4}$$

and the boundary conditions are,

$$\begin{aligned} u(0,x_2,t) - 2u_{x_1}(0,x_2,t) &= g_0(x_2,t), \quad u_{x_1}(1,x_2,t) = g_1(x_2,t), \\ u(x_1,0,t) - 2u_{x_1}(x_1,0,t) &= h_0(x_1,t), \quad u_{x_2}(x_1,0,t) = h_1(x_1,t). \end{aligned}$$

The functions f g_0, g_1 h_0 and h_1 are chosen such that the solution is

$$(e^{-t}+1)(e^{x_1} + 2e^{x_2}).$$

```
************************************************************************
C     PROGRAM MAIN

      DOUBLE PRECISION LX, DLX, LY, DLY, ONE, ZERO, HUDRD
      PARAMETER (ONE = 1.D0, ZERO=0.D0, HUDRD=100.D0)
      PARAMETER( LX = ONE, LY = ONE )

      INTEGER KXTOP, KYTOP, NBLTOX, NBLTOY, M, NRWORK, NIWORK,
     *   NRPAR, NIPAR, NSLT02, NP, INCX

      PARAMETER(M=2, KXTOP=2, KYTOP=2, NBLTOX=32, NBLTOY=32)

      INTEGER BWORKX, BWORKY, NKSXP2

C     THE SIZE OF WORK ARRAY ETC.
      PARAMETER(BWORKX = ((M+KXTOP+1)*(M+KXTOP))/2+(M+1)*(M+KXTOP),
     *          BWORKY = ((M+KYTOP+1)*(M+KYTOP))/2+(M+1)*(M+KYTOP),
     *          NKSXP2 = KXTOP*NBLTOX+2,
     *          NSLT02 = NKSXP2*(KYTOP*NBLTOY+2))

      PARAMETER(NIWORK = 20+KXTOP*NBLTOX*KYTOP*NBLTOY,
     *          NIPAR  = 100+3*NBLTOY+3*NBLTOX+2*KXTOP*NBLTOX+
     *                   2*KYTOP*NBLTOY,
     *          NRWORK = 42+11*KXTOP*NBLTOX*KYTOP*NBLTOY-2*KXTOP*
     *                   NBLTOX*KYTOP+KXTOP*NBLTOX*KYTOP**2*NBLTOY,
     *          NRPAR  = 84+31*KXTOP*NBLTOX+17*KYTOP*NBLTOY+
     *                   KXTOP**2*NBLTOX**2+((81+17*KXTOP)*KXTOP)/2+
     *                   NBLTOX+NBLTOY+2*KXTOP*NBLTOX*KYTOP*NBLTOY+
     *                   11*KXTOP**2*NBLTOX+((7+KYTOP)*KYTOP)/2+
     *                   5*KYTOP**2*NBLTOY)

C     DECLARE THE WORK ARRAY ETC.
      DOUBLE PRECISION XI(NBLTOX+1), YI(NBLTOY+1), RWORK(NRWORK),
     *   RPAR(NRPAR), Y(NSLT02), YPRIME(NSLT02), T, TOUT, TOL,
     *   CONTX(8), CONTY(8), RTOL, ATOL
      PARAMETER ( TOL=1.0D-10, INCX=1 )
      INTEGER IWORK(NIWORK), IPAR(NIPAR), KX, NBLOCX, KY, NBLOCY,
     *   TDEPCX, TDEPCY, BFLAG, TBFLAG, NYDIM, IN1, IN2, IEVAL
      DOUBLE PRECISION PAX, PCX, PAY, PBY, PCY, FUNINT, FUNRIG,
     *   PG0, PG1, PH0, PH1, TPG0, TPG1, TPH0, TPH1, PHODX, PH1DX,
     *   PGODY, PG1DY, PHODTX, PH1DTX, PGODTY, PG1DTY, A0, B0, A1,
     *   B1, R0, D0, R1, D1
      COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
      EXTERNAL PAX, PCX,PAY, PBY, PCY, FUNINT, FUNRIG, PG0, PG1,
     *   PH0, PH1, TPG0, TPG1, TPH0, TPH1, PHODX, PH1DX, PGODY,
     *   PG1DY, PHODTX, PH1DTX, PGODTY, PG1DTY, DRES, DJAC

C     FOR EVALUATION OF THE SOLUTION.
      DOUBLE PRECISION WORKX(BWORKX), WORKY(BWORKY), WORKXY(NKSXP2),
     *   XPOINT(200), YPOINT(200), SARRAY(200), DXARAY(200),
     *   DYARAY(200), LINFER, LINFSL, EXACTP, EXACT, DTEXAT, TEMP1,
     *   TEMP2, FLOAT, ABS
      INTRINSIC FLOAT, ABS
```

```
C     INPUT DATA.
      READ(5,*) TDEPCX, TDEPCY
      READ(5,*) AO, BO, A1, B1
      READ(5,*) RO, DO, R1, D1
      READ(5,*)KX, NBLOCX
      READ(5,*)KY, NBLOCY
      READ(5,*)TOUT
      RTOL = ZERO
      READ *,ATOL
      IF(ATOL .LE. ZERO) THEN
         ATOL = TOL
      ENDIF

C     SETUP FOR THE TOTAL NUMBER OF UNKNOWNS.
      NYDIM = (KX*NBLOCX+2)*(KY*NBLOCY+2)

C     USE FINITE DIFFERENCE FOR THE DERIVATIVE WITH RESPECT TO T
C     OF THE BOUNDARY FUNCTIONS.
      BFLAG = 0
      TBFLAG = 0

C     THE INITIAL VALUE OF TIME VARIABLE IS ZERO.
      T = ZERO

C     GET THE STEP SIZE.
      DLX = LX/FLOAT(NBLOCX)
      DLY = LY/FLOAT(NBLOCY)

C     SET UP MESH.
      DO 2 IN1=1,NBLOCX+1
         XI(IN1) = (FLOAT(IN1)-ONE)*DLX
    2 CONTINUE

      DO 3 IN1=1,NBLOCY+1
         YI(IN1) = (FLOAT(IN1)-ONE)*DLY
    3 CONTINUE

      CALL PARCOL(KX, NBLOCX, KY, NBLOCY, TDEPCX, TDEPCY,
     1   T, TOUT, PAX, PCX, PAY, PBY, PCY, FUNINT, FUNRIG,
     2   XI, YI, AO, BO, A1, B1, RO, DO, R1, D1, PHO, PH1,
     3   PGO, PG1, BFLAG, PHODX, PH1DX, PGODY, PG1DY, TPHO,
     4   TPH1, TPGO, TPG1, TBFLAG, PHODTX, PH1DTX, PGODTY,
     5   PG1DTY, RTOL, ATOL, Y, YPRIME, NYDIM, CONTX, CONTY,
     6   DRES, DJAC, RWORK, NRWORK, RPAR, NRPAR, IWORK,
     7   NIWORK, IPAR, NIPAR)

      WRITE(6,*)'T AND TOUT ARE ', T, TOUT
      WRITE(6,*)'KX, NBLOCX ARE ', KX, NBLOCX
      WRITE(6,*)'KY, NBLOCY ARE ', KY, NBLOCY
      WRITE(6,*)'AO, BO, A1, B1 ', AO, BO, A1, B1
      WRITE(6,*)'RO, DO, R1, D1 ', RO, DO, R1, D1
```

```
C     SETUP THE X VALUES AT WHICH THE SOLUTION IS DESIRED.
      NP = NBLOCX+1
      DO 100 IN1=1,NBLOCX+1
         XPOINT(IN1) = XI(IN1)
  100 CONTINUE

C     SETUP FOR COMPUTATION OF THE MAXIMUM ERROR AT MESH POINTS.
      LINFER = ZERO
      LINFSL = ZERO

C     EVALUATE THE SOLUTION AND FIND THE MAXIMUM ERROR AT MESH
C     POINTS.
      IEVAL = 1
      DO 170 IN2=1,NBLOCY+1
         TEMP1 = YI(IN2)
         CALL EVALSN(XPOINT, SARRAY, DXARAY, DYARAY,
     *      TEMP1, Y, XI, YI, NP, NBLOCX, KX,
     *      NBLOCY, KY, CONTX, CONTY,
     *      WORKX, WORKY, WORKXY, IEVAL)
         DO 160 IN1=1,NBLOCX+1
            EXACTP = EXACT(XPOINT(IN1),TEMP1,T)
            IF ( LINFER .LT. ABS(EXACTP - SARRAY(IN1)) ) THEN
               LINFER = ABS(EXACTP - SARRAY(IN1))
            ENDIF
            IF ( LINFSL .LT. ABS(EXACTP) ) LINFSL=ABS(EXACTP)
  160    CONTINUE
  170 CONTINUE

C     OUTPUT THE MAXIMUM ERROR AT MESH POINTS.
      PRINT *,'NODE    ERROR       = ', LINFER
      PRINT *,'LINFTY  NORM        = ', LINFSL

C     SETUP THE SAMPLE POINTS FOR THE L-INFINITY ERROR.
      DLX = LX/HUDRD
      DLY = LY/HUDRD
      DO 250 IN1=1,101
         XPOINT(IN1) = FLOAT((IN1-1)*DLX)
         YPOINT(IN1) = FLOAT((IN1-1)*DLY)
  250 CONTINUE

C     SETUP ZERO FOR THE L-INFINITY ERROR.
      LINFER = ZERO
      LINFSL = ZERO

C     EVALUATE THE SOLUTION AND FIND THE L-INFINITY ERROR.
      NP = 101
      DO 180 IN2=1,101
         TEMP1 = YPOINT(IN2)
         CALL EVALSN(XPOINT, SARRAY, DXARAY, DYARAY,
     *      TEMP1, Y, XI, YI, NP, NBLOCX, KX,
     *      NBLOCY, KY, CONTX, CONTY,
     *      WORKX, WORKY, WORKXY, IEVAL)
         DO 190 IN1=1,101
```

```
            EXACTP = EXACT(XPOINT(IN1),TEMP1,T)
            TEMP2 = ABS(EXACTP - SARRAY(IN1))
            IF ( LINFER .LT. TEMP2 ) THEN
               LINFER = TEMP2
            ENDIF
            IF ( LINFSL .LT. ABS(EXACTP) ) LINFSL=ABS(EXACTP)
  190    CONTINUE
  180 CONTINUE

C     OUTPUT THE L-INFINITY ERROR.
      PRINT *,'UNIFORM   ERROR     = ', LINFER
      PRINT *,'LINFTY    NORM      = ', LINFSL

99999 STOP
      END
```

The following functions define the operators L_1 and L_2.

```
      DOUBLE PRECISION FUNCTION PAX(X,T)
      DOUBLE PRECISION X, T, ONE, TWO
      PARAMETER ( ONE = 1.0D0, TWO=2.0D0 )
      PAX = (T+ONE)*(X**2+TWO)
      RETURN
      END

      DOUBLE PRECISION FUNCTION PCX(X,T)
      DOUBLE PRECISION X, T, PI, ONE, SIN, ACOS
      INTRINSIC SIN
      PARAMETER ( ONE = 1.0D0 )
      PI = ACOS(-ONE)
      PCX = T*SIN(PI*X)
      RETURN
      END


      DOUBLE PRECISION FUNCTION PAY(X,T)
      DOUBLE PRECISION X, T, TWO, FIVE
      PARAMETER ( TWO=2.0D0, FIVE=5.0D0 )
      PAY = (T+TWO)*(X**4+FIVE)
      RETURN
      END

      DOUBLE PRECISION FUNCTION PBY(X,T)
      DOUBLE PRECISION X, T, PI, ONE, SIN, ACOS
      INTRINSIC SIN, ACOS
      PARAMETER ( ONE = 1.0D0 )
      PI = ACOS(-ONE)
      PBY = SIN(PI*X)+T
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION PCY(X,T)
      DOUBLE PRECISION X, T, ONE, PI, COS, ACOS
      DOUBLE PRECISION COS, ACOS
      PARAMETER ( ONE = 1.0D0 )
      PI = ACOS(-ONE)
      PCY = COS(PI*X)*T**2
      RETURN
      END
```

The following function gives the initial value function.

```
      DOUBLE PRECISION FUNCTION FUNINT(X,Y)
      DOUBLE PRECISION EXP, X, Y, ONE, TWO
      INTRINSIC EXP
      PARAMETER (ONE=1.0D0,TWO=2.0D0)
      FUNINT = TWO*(EXP(X)+TWO*EXP(Y))
      RETURN
      END
```

The following function gives the right hand side function of the differential equation.

```
      DOUBLE PRECISION FUNCTION FUNRIG(X,Y,T)
      DOUBLE PRECISION EXP, X, Y, T, ONE, TWO, FIVE, PI,
     *   SIN, COS, ACOS
      INTRINSIC SIN, COS, ACOS, EXP
      PARAMETER (ONE=1.0D0,TWO=2.0D0, FIVE=5.0D0 )
      PI = ACOS(-ONE)
      FUNRIG = -EXP(-T)*(EXP(X)+TWO*EXP(Y))-
     *   (T+ONE)*(X**2+TWO)*(EXP(-T)+ONE)*EXP(X)-
     *   T*SIN(PI*X)*(EXP(-T)+ONE)*(EXP(X)+TWO*EXP(Y))-
     *   TWO*(T+TWO)*(Y**4+FIVE)*(EXP(-T)+ONE)*EXP(Y)-
     *   TWO*(SIN(PI*Y)+T)*(EXP(-T)+ONE)*EXP(Y)-
     *   COS(PI*Y)*T**2*(EXP(-T)+ONE)*(EXP(X)+TWO*EXP(Y))
      RETURN
      END
```

The following functions define the boundary conditions.

```
      DOUBLE PRECISION FUNCTION PGO(X, T)
      DOUBLE PRECISION X, T, ZERO, XBNDRY
      PARAMETER ( ZERO = 0.0D0 )
      DOUBLE PRECISION AO, BO, A1, B1, RO, DO, R1, D1
      COMMON /BCNDS/ AO, BO, A1, B1, RO, DO, R1, D1
      PGO = XBNDRY(AO,BO,ZERO,X,T)
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION PG1(X, T)
      DOUBLE PRECISION X, T, ONE, XBNDRY
      PARAMETER ( ONE = 1.0D0 )
      DOUBLE PRECISION A0, B0, A1, B1, R0, D0, R1, D1
      COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
      PG1 = XBNDRY(A1,B1,ONE,X,T)
      RETURN
      END

      DOUBLE PRECISION FUNCTION PH0(X, T)
      DOUBLE PRECISION X, T, ZERO, YBNDRY
      PARAMETER ( ZERO = 0.0D0 )
      DOUBLE PRECISION  A0, B0, A1, B1, R0, D0, R1, D1
      COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
      PH0 = YBNDRY(R0,D0,X,ZERO,T)
      RETURN
      END

      DOUBLE PRECISION FUNCTION PH1(X, T)
      DOUBLE PRECISION X, T, ONE, YBNDRY
      PARAMETER ( ONE = 1.0D0 )
      DOUBLE PRECISION  A0, B0, A1, B1, R0, D0, R1, D1
      COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
      PH1 = YBNDRY(R1,D1,X,ONE,T)
      RETURN
      END

      DOUBLE PRECISION FUNCTION XBNDRY(AU,AUX,X,Y,T)
      DOUBLE PRECISION AU,AUX,X,Y,T,EXACT,DXEXAT
      XBNDRY = AU*EXACT(X,Y,T) - AUX*DXEXAT(X,Y,T)
      RETURN
      END

      DOUBLE PRECISION FUNCTION YBNDRY(AU,AUY,X,Y,T)
      DOUBLE PRECISION AU,AUY,X,Y,T,EXACT,DYEXAT
      YBNDRY = AU*EXACT(X,Y,T) - AUY*DYEXAT(X,Y,T)
      RETURN
      END
```

The following functions give the derivatives of the boundary conditions with respect to t.

```
      DOUBLE PRECISION FUNCTION TPG0(X, T)
      DOUBLE PRECISION X, T, ZERO, TXBNRY
      PARAMETER ( ZERO = 0.0D0 )
      DOUBLE PRECISION A0, B0, A1, B1, R0, D0, R1, D1
      COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
      TPG0 = TXBNRY(A0,B0,ZERO,X,T)
      RETURN
      END
```

```
DOUBLE PRECISION FUNCTION TPG1(X, T)
DOUBLE PRECISION X, T, ONE, TXBNRY
PARAMETER ( ONE = 1.0D0 )
DOUBLE PRECISION A0, B0, A1, B1, R0, D0, R1, D1
COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
TPG1 = TXBNRY(A1,B1,ONE,X,T)
RETURN
END

DOUBLE PRECISION FUNCTION TPH0(X, T)
DOUBLE PRECISION X, T, ZERO, TYBNRY
PARAMETER ( ZERO = 0.0D0 )
DOUBLE PRECISION  A0, B0, A1, B1, R0, D0, R1, D1
COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
TPH0 = TYBNRY(R0,D0,X,ZERO,T)
RETURN
END

DOUBLE PRECISION FUNCTION TPH1(X, T)
DOUBLE PRECISION X, T, ONE, TYBNRY
PARAMETER ( ONE = 1.0D0 )
DOUBLE PRECISION  A0, B0, A1, B1, R0, D0, R1, D1
COMMON /BCNDS/ A0, B0, A1, B1, R0, D0, R1, D1
TPH1 = TYBNRY(R1,D1,X,ONE,T)
RETURN
END

DOUBLE PRECISION FUNCTION TXBNRY(AU,AUX,X,Y,T)
DOUBLE PRECISION AU,AUX,X,Y,T,DTEXAT,DXTEXA
TXBNRY = AU*DTEXAT(X,Y,T) - AUX*DXTEXA(X,Y,T)
RETURN
END

DOUBLE PRECISION FUNCTION TYBNRY(AU,AUY,X,Y,T)
DOUBLE PRECISION AU,AUY,X,Y,T,DTEXAT,DYTEXA
TYBNRY = AU*DTEXAT(X,Y,T) - AUY*DYTEXA(X,Y,T)
RETURN
END
```

The following functions give the solution and the derivatives of the solution.

```
DOUBLE PRECISION FUNCTION EXACT(X,Y,T)
DOUBLE PRECISION EXP, X, Y, T, ONE, TWO
INTRINSIC EXP
PARAMETER ( ONE = 1.0D0,TWO=2.0D0 )
EXACT = (EXP(-T)+ONE)*(EXP(X)+TWO*EXP(Y))
RETURN
END

DOUBLE PRECISION FUNCTION DXEXAT(X,Y,T)
DOUBLE PRECISION EXP, X, Y, T, ONE
INTRINSIC EXP
```

```
      PARAMETER ( ONE = 1.0D0 )
      DXEXAT = (EXP(-T)+ONE)*EXP(X)
      RETURN
      END

      DOUBLE PRECISION FUNCTION DYEXAT(X,Y,T)
      DOUBLE PRECISION EXP, X, Y, T, ONE, TWO
      INTRINSIC EXP
      PARAMETER ( ONE = 1.0D0,TWO=2.0D0 )
      DYEXAT = (EXP(-T)+ONE)*TWO*EXP(Y)
      RETURN
      END
```

The following functions need to be supplied as below. Since we use finite differences to approximate the derivatives, they are not called in this sample program.

```
      DOUBLE PRECISION FUNCTION PGODY()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PG1DY()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PHODX()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PH1DX()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PGODTY()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PG1DTY()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PHODTX()
      RETURN
      END

      DOUBLE PRECISION FUNCTION PH1DTX()
      RETURN
      END
```

Some sample input data:

```
0 0
1.D0  2.D0 0.D0 -1.D0
2.D0 -5.D0 3.D0 -4.D0
2 32
2 32
1.D0
1.D-7
```

The corresponding output:

```
T AND TOUT ARE      1.0    1.0
KX, NBLOCX ARE      2  32
KY, NBLOCY ARE      2  32
AO, BO, A1, B1      1.0    2.0   0.0  -1.0
RO, DO, R1, D1      2.0   -5.0   3.0  -4.0
NODE    ERROR       =     7.8227486710603D-09
LINFTY  NORM        =     11.154845485377
UNIFORM   ERROR     =     2.4945705945356D-08
LINFTY    NORM      =     11.154845485377
```

Bibliography

[1] J. Adams, P. Swarztrauber, and R. Sweet, *FISHPACK: Efficient FORTRAN subprograms for the solution of separable elliptic partial differential equations, Version 3*, National Center for Atmospheric Research, Boulder, Colorado, 1978.

[2] U. Ascher, *Collocation for two-point boundary value problems revisited*, SIAM J. Numer. Anal., Vol. 23 (1986), pp. 596-609.

[3] U. Ascher, J. Christiansen, and R.D. Russell, *A collocation solver for mixed order systems of boundary value problems*, Math. Comp., Vol. 33 (1979), pp. 659-679.

[4] U. Ascher, J. Christiansen, and R.D. Russell, *Collocation Software for Boundary-value ODEs*, ACM Trans. Math. Software, Vol. 7 (1981), pp. 209-222.

[5] U. Ascher, S. Pruess, and R.D. Russell, *On spline basis selection for solving differential equations*, SIAM J. Numer. Anal., Vol. 20 (1983), pp. 121-142.

[6] U. Ascher and R.D. Russell, *Evaluation of B-splines for solving systems of boundary value problems*, Dept. of Comp. Sci. Tech. Report 77-14, University of British Columbia, Vancouver, British Columbia, 1977.

[7] K.R. Bennett, *Parallel Collocation Methods for Boundary Value Problems*, Ph.D. thesis, University of Kentucky, Lexington, Kentucky, 1991.

[8] K.R. Bennett and G. Fairweather, *A parallel boundary value ODE code for shared-memory machines*, Int. J. High Speed Computing, Vol. 4 (1992), pp. 71-86.

[9] M. Berzins and R.M. Furzeland, *A user's manual for SPRINT-A versatile software package for solving systems of algebraic, ordinary and partial differential equations:*

Part 1-algebraic and ordinary differential equations, Tech. Report 85.058, Thornton Research Center, Chester, England, 1985.

[10] B. Bialecki and X.C. Cai, *H^1-norm error bounds for piecewise Hermite bicubic orthogonal spline collocation schemes for elliptic boundary value problems*, SIAM J. Numer. Anal., to appear.

[11] B. Bialecki and G. Fairweather, *Matrix decomposition algorithms in orthogonal spline collocation for separable elliptic boundary value problems*, SIAM J. Sci. Comput., to appear.

[12] C. de Boor and B. Swartz, *Collocation at Gaussian points*, SIAM J. Numer. Anal., Vol. 10 (1973), pp. 582-606.

[13] C. de Boor, *Package for calculating with B-splines*, SIAM J. Numer. Anal., Vol. 14 (1977), pp. 441-472.

[14] C. de Boor, *A Practical Guide to Splines*, Applied Math. Science, Vol. 27, Springer-Verlag, New York, 1978.

[15] K.E. Brenan, S.L. Campbell, and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland, New York, 1989.

[16] J. Cerutti and S.V. Parter, *Collocation methods for parabolic differential equations in one space dimension*, Numer. Math., Vol. 26 (1976), pp. 227-254.

[17] C.R. Crawford, *Reduction of a band-symmetric generalized eigenvalue problem*, Commun. ACM, Vol. 16 (1973), pp. 41-44.

[18] P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Second Edition, Academic Press, New York, 1984.

[19] J.C. Diaz, G. Fairweather, and P. Keast, *FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, Vol. 9 (1983), pp. 358-375.

[20] J.C. Diaz, G. Fairweather, and P. Keast, *Algorithm 603: COLROW and ARCECO: FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, Vol. 9 (1983), pp. 376-380.

[21] J.J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users' Guide*, SIAM Pub., Philadelphia, 1979.

[22] J.J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, *An extended set of FORTRAN basic linear algebra subprograms*, ACM Trans. Math. Software, Vol. 14 (1988), pp. 1-17.

[23] J.J. Dongarra, J.Du Croz, S. Hammarling, and R.J. Hanson, *Algorithm 656 An extended set of FORTRAN basic linear algebra subprograms: Model implementation and test program*, ACM Trans. Math. Software, Vol. 14 (1988), pp. 18-32.

[24] J. Douglas Jr. and T. Dupont, *Collocation Methods for Parabolic Equations in a Single Space Variable*, Lecture Notes in Mathematics 385, Spring-Verlag, New York, 1974.

[25] G. Fairweather and D. Meade, *A survey of spline collocation methods for the numerical solutions of differential equations*, Mathematics for Large Scale Computing, Marcel Dekker, New York, Vol. 120 (1989), pp. 297-341.

[26] B.S. Garbow, J.M. Boyle, J.J. Dongarra and C.B. Moler, *Matrix Eigensystems Routines-EISPACK Guide Extension*, Lecture Notes in Computer Science 51, Spring-Verlag, New York, 1977.

[27] C.W. Gear, *The simultaneous numerical solution of differential/algebraic equations*, IEEE Trans. Circuit Theory, CT-18 (1971), pp. 89-95.

[28] C.W. Gear and L.R. Petzold, *ODE methods for the solution of differential/algebraic systems*, SIAM J. Numer. Anal., Vol. 21 (1984), pp. 716-728.

[29] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Second Edition, The John Hopkins University Press, Baltimore, Maryland, 1989.

[30] S.L. Graham, P.B. Kessler and M.K. McKusick, *'gprof: A Call Graph Execution Profiler*, Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, (1982), pp. 120-126.

[31] A.C. Hindmarsh, *GEAR: Ordinary differential equation system solver*, Report UCID-30001, Lawrence Livermore Laboratory, Livermore, California, 1974.

[32] A.C. Hindmarsh and G.D. Byrne, *EPISODE: An experimental package for the integration of system of ordinary differential equations*, Report UCID-16577, Lawrence Livermore Laboratory, Livermore, California, 1974.

[33] A.C. Hindmarsh, *Preliminary documentation of GEARIB: Solution of implicit systems of ODE with banded Jacobian*, Report UCID-30130, Lawrence Livermore Laboratory, Livermore, California, 1976.

[34] A.C. Hindmarsh and G.D. Byrne, *Application of EPISODE: An experimental package for the integration of system of ordinary differential equations*, Numerical Methods for Differential Systems, L. Lapidus and W.E. Schiesser (eds), Academic Press, New York, 1976, pp. 147-166.

[35] A.C. Hindmarsh, *LSODE and LSODI, Two new initial value ordinary differential equation solvers*, ACM-SIGNUM Newsletter, Vol. 15 (1980) pp. 10-11.

[36] A.C. Hindmarsh, *ODEPACK-A systematized collocation of O.D.E. solvers*, Scientific Computing, R. Stepleman et al.(eds), North-Holland, New York, 1983, pp. 55-64.

[37] E.N. Houstis, W.F. Mitchell, and J.R. Rice, *Collocation software for second-order elliptic partial differential equations*, ACM Trans. Math. Software, Vol. 11 (1985), pp. 379-412.

[38] J. Hornsby, *EPDE1-A computer program for the solution of elliptic partial differential equations(potential problems)*, Computer Center Program Library Long Write-Up D-300, CERN, Geneva, 1977.

[39] K.R. Jackson and R.N. Pancer, *The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs*, Comp. Sci. Tech. Rep. No. 255, University of Toronto, Toronto, Ontario, 1991.

[40] L. Kaufman, *Banded eigenvalue solvers on vector machines*, ACM Trans. Math. Software, Vol. 10 (1984), pp. 73-86.

[41] L. Kaufman and D.D. Warner, *Algorithm 685: A program for solving separable elliptic equations*, ACM Trans. Math. Software, Vol. 16 (1990), pp. 325-351.

[42] P. Keast, G. Fairweather, and J.C. Diaz, *A computational study of finite element methods for second order linear two-point boundary value problems*, Math. Comp., Vol. 40 (1981), pp. 499-518.

[43] P. Keast and P.H. Muir, *Algorithm 688: EPDCOL: A more efficient PDECOL code*, ACM Trans. Math. Software, Vol. 17 (1991), pp. 153-166.

[44] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T..Krogh, *Basic linear algebra subprograms for FORTRAN usage*, ACM Trans. Math. Software, Vol. 5 (1979), pp. 308-323.

[45] C.L. Lawson, R.J. Hanson, D.R. Kincaid and F.T. Krogh, *Algorithm 539: Basic linear algebra subprograms for FORTRAN usage [F1]* , ACM Trans. Math. Software, Vol. 5 (1979), pp. 324-325.

[46] J.M. Levesque and J.W. Williamson, *A Guidebook to FORTRAN on Supercomputers*, Academic Press, London, 1988.

[47] O.A. Liskovets, *The methods of lines (review)*, Differential Equations, Vol. 1 (1965), pp. 1308-1323.

[48] R.E. Lynch, J.R. Rice, and D.H. Thomas, *Direct solution of partial differential equations by tensor product methods*, Numer. Math., Vol. 4 (1964), pp. 185-199.

[49] *NAG FORTRAN Library Manual*, Mark 15, Vol. 5, NAG Ltd., London, 1991.

[50] M. Marchura and R..A. Sweet, *A survey of software for partial differential equations*, ACM Trans. Math. Software, Vol. 6 (1980), pp. 461-488.

[51] N.K.Madsen and R.F. Sincovec, *Algorithm 540: PDECOL, General collocation software for partial differential equations*, ACM Trans. Math. Software, Vol. 5 (1979), pp. 326-351.

[52] F. Majaess, P. Keast, G. Fairweather, and K.R. Bennett, *Algorithm 704: ABDPACK and ABBPACK-FORTRAN programs for the solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions*, ACM Trans. Math. Software, Vol. 18 (1992), pp. 205-210.

[53] M. Marcus and H. Minc, *A Survey of Matrix Theory and Matrix Inequalities*, Allyn and Bacon, Boston, 1964.

[54] R.S. Martin and J.H. Wilkinson, *Reduction of the symmetric eigenvalues problem* $A\vec{x} = \lambda\vec{x}$ *and related problems to standard form*, Numer. Math., Vol. 11 (1968), pp. 99-110.

[55] P.K. Moore and J.E. Flaherty, *High-order adaptive finite element-singly implicit Runge-Kutta methods for parabolic differential equations*, BIT, Vol. 33 (1993), pp.309-331.

[56] J.M. Ortega and R.G. Voigt, *Solution of parallel differential equations on vector and parallel computers*, SIAM Review, Vol. 27 (1985), pp. 149-240.

[57] M. Paprzycki and I. Gladwell, *Solving almost block diagonal system on parallel computers*, Parallel Computing, Vol. 17 (1991), pp. 133-153.

[58] G. Peters and J.H. Wilkinson, $Ax = \lambda Bx$ *and the generalized eigenproblem*, SIAM J. Numer. Anal., Vol. 7 (1970), pp. 479-492.

[59] L.R. Petzold, *A Description of DASSL: A differential/algebraic system solver*, Scientific Computing, R. Stepleman et al.(eds), North-Holland, New York, 1983, pp. 65-68.

[60] L.R. Petzold, *Differential/algebraic equations are not ODEs*, SIAM J. Sci. Stat. Comput., Vol. 3 (1982), pp. 367-384.

[61] L.R. Petzold, *Order results for implicit Runge-Hutta methods applied to differential/algebraic systems*, SIAM J. Numer. Anal., Vol 23 (1986), pp. 837-852.

[62] J.C. Pirkle, Jr. and W.E. Schiesser, *DSS/2: A transportable FORTRAN 77 code for systems of ordinary and one, two and three-dimensional partial differential equations*, manuscript, 1987 Summer Computer Simulation Conference, Montreal, 1987.

[63] J. Rice, *ELLPACK: A research tool for elliptic partial differential equation software*, Mathematical Software III, Academic Press, New York, 1977, pp. 319-341.

[64] J. Roux, J. Tourneur and D. Vandorpe, *Automatic generation of partial differential equations integration programs*, Advances in Computer Methods for Partial Differential Equations-II, R. Vichnevetsky (Ed.), IMACS (AICA), Rutgers Univ., New Brunswick, New Jersey, 1977.

[65] W.E. Schiesser, *DSS/2(Differential systems simulator, version 2), introductory programming manual for ordinary differential equations*, manuscript, 1976.

[66] P.N. Swarztrauber and R.A. Sweet, *Vector and parallel methods for the direct solution of Poisson's equation*, J. Comp. Appl. Math., Vol. 27 (1989), pp. 241-263.

[67] J. Taylor and J. Taylor *ELIPTI-TORMAC: A code for the solution of general nonlinear elliptic problems over 2-D regions of arbitrary shape*, Advances in Computer Methods for partial differential equations-II, R. Vichnevetsky (Ed.), IMACS (AICA), Rutgers Univ., New Brunswick, New Jersey, 1977.

[68] C. Thomas, *POTENT-A package for the numerical solution of potential problems in general two-dimensional regions*, Software for Numerical Mathematics, D. Evans (Ed.), Academic Press, New York, 1974, pp. 315-336.

[69] R.C. Thompson, *Convergence and error estimates for the method of lines for certain nonlinear elliptic and elliptic-parabolic equations*, SIAM J. Numer. Anal., Vol. 13 (1976), pp. 27-43.

[70] G.M. Vainikko, *Convergence of the collocation method for nonlinear differential equations*, USSR Comp. Math. Phys., Vol. 6 (1966), pp. 47-58.

[71] M. Vajteršic, *Parallel marching Poisson solvers*, Parallel Computing, Vol. 1 (1984), pp. 325-330.

[72] S.J. Wright, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Comput., Vol. 13 (1992), pp. 742-764.

[73] S.J. Wright, *Stable parallel elimination for boundary value ODEs*, Numer. Math., to appear.