MASTER'S THESIS

An Object-Oriented Programming Approach to Implement Global Spectral Methods: Application to Dynamic Simulation of a Chemical Infiltration Process

by Jiefei Huang
Advisor: Raymond A. Adomaitis

M.S. 2000-1

INSTITUTE FOR SYSTEMS RESEARCH

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

ABSTRACT

Title of Thesis:	AN OBJECT-ORIENTED PROGRAMMING APPROACH TO IMPLEMENT GLOBAL SPECTRAL METHODS: APPLICATION TO DYNAMIC SIMULATION OF A CHEMICAL VAPOR INFILTRATION PROCESS
Degree candidate:	Jiefei Huang
Degree and year:	Master of Science, 2000
Thesis directed by:	Professor Raymond A. Adomaitis Department of Chemical Engineering

Boundary-value problems (BVPs) in relatively simple geometries can be solved using global spectral methods. These discretization methods are applicable to a wide range of problems and are suitable for a "rapid prototyping" approach to simulator development for complex systems. Object-Oriented Programming techniques for solving BVPs are introduced in this work. Object classed are created to encapsulate trial function sequences, discretized differential and quadrature operators, and other data structures used for spectral discretization and projection operations. Operator/function overloading subsequently is used to numerically implement the Galerkin projection method. Emphasis is placed on developing numerical methods suitable for discretizing 2- and 3-dimensional problems, integrating the resulting ODE/AE systems in time, and reconstructing the solutions in the physical space. A detailed model of an isothermal carbon-carbon chemical vapor infiltration (CVI) system was studied as a true test of the ability of the numerical methods.

AN OBJECT-ORIENTED PROGRAMMING APPROACH TO IMPLEMENT GLOBAL SPECTRAL METHODS: APPLICATION TO DYNAMIC SIMULATION OF A CHEMICAL VAPOR INFILTRATION PROCESS

by

Jiefei Huang

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2000

Advisory Committee:

Professor Raymond A. Adomaitis, Chair/Advisor
Professor James W. Gentry
Professor Michael T. Harris

© Copyright by

Jiefei Huang

2000

DEDICATION

To My Parents

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Introduction to the Spectral Methods

Spectral methods involve representing the solution to a differential equation model in terms of a truncated series of known smooth, global, orthogonal trial functions of the independent variables. Spectral methods have been utilized in many areas, with the major emphasis occurring in two: global atmospheric modeling and fundamental turbulence studies. Most applications have been to time-dependent mixed initial-boundary-value problems with finite-difference or other time-stepping schemes to provide integration in time.

The choice of an appropriate spectral method is governed by two main considerations [11]:

(1) **Accuracy**. A spectral method should be designed to give results of greater accuracy than can be obtained by more conventional difference methods using similar spatial resolution or degrees of freedom. The choice of appropriate spectral representation depends on the kind of boundary conditions involved in the problem.

(2) **Efficiency**. The spectral methods should be at least as efficient as dif-

ference methods with comparable numbers of degrees of freedom. Therefore, for similar amounts of work, the spectral methods should produce more accurate results than a finite-difference method.

The accuracy and efficiency of a spectral method depends on making the correct choice for the test and trial functions. The proper choice depends on the problem and on the nature of the boundary conditions. When an appropriate choice is made, spectral methods provide a very high rate of convergence once sufficient trial functions are included to adequately represent the underlying problem. If an inappropriate choice is made, the result can be poor accuracy or even convergence to a spurious solution [11].

The finite-difference formulation may appear to offer a more direct approach to the numerical solution of partial differential equations than does a spectral method. It simply replaces the derivatives with finite-difference expansions and demands that the resulting algebraic equations be satisfied exactly at the grid points. However, difficulties can arise in imposing boundary conditions, and low-order finite-difference formulations are often inaccurate, particularly on a coarse grid.

Spectral discretization methods do, however, have some drawbacks: they can be difficult to code and may be inflexible compared to finite-difference or finite-element methods. The finite-difference methods require relatively little algebraic manipulation and relatively straightforward programming. Finite-element methods require some preliminary algebraic manipulation and more programming effort than finite-difference methods. However, the modularity of the finite-element method lends itself to efficient programming. In solving a new problem, relatively few changes need be made in an existing computer package. Spectral methods require substantial preliminary algebraic manipulation and programming if an efficient code is to be generated. Also, the solution of a new problem typically requires a new set of trial functions, new boundary-condition specification, and so on, in short, a complete new program.

Therefore, our intention was to develop a computational toolbox consisting of a common set of numerical tools for implementing the different spectral methods inside the Matlab computational environment.

1.2 Motivation and Goals of This Research

Boundary-value problems in relatively simple geometries can be solved using global spectral methods. These discretization methods are applicable to a wide range of problems and are suitable for a "rapid prototyping" approach to simulator development for complex systems. These methods can be extended to systems defined in complex geometries though the spectral element methods or by replacing boundary conditions with forcing functions to the BVPs. We focus on the development of numerical methods for high- dimensional systems, especially 2- or 3- dimensional systems in the physical domain. These methods extend the quadrature-based MWR methods presented in Lin and co-workers [13]. The main issues addressed in this work are listed below.

Object-Oriented programming Object-Oriented Programming techniques for MWR are introduced in this work. Related variables, functions are encapsulated in "classes". Operator/function overloading is used to make the operations more flexible. This will make programming more straightforward and the programs easier to manipulate and maintain.

High-dimensional projections Numerical techniques for implementing projection methods on 2- or 3- dimensional systems defined in the physical domain and for efficiently reconstructing state variable profiles from n-dimensional trial function expansions are developed.

Heterogeneous systems A more coherent approach to working with heterogeneous systems (systems defined by multiple BVPs on one or more spatial domains) is developed. This approach is based on defining the solution in a Matlab cell array structure. The residual functions and associated Jacobian arrays are likewise arranged. This meant that matrix multiplication, methods for solving sets of equations defined in this cell array structure, and other analogs to linear algebra operations had to be developed for cell arrays that could contain n-dimensional arrays.

ODE/AE systems A numerical integrator for these cell structures is created. In particular, a method for solving the ODE/AE systems that result from the semi-discrete projection methods is needed. Time is treated differently from the other distributions because the time intervals over which all the state variables are defined are the same. In this thesis work, implementing nonlinear Galerkin projections and other advanced MWR also are investigated.

1.3 Modeling of a CVI Process

Composite materials are increasingly being regarded as the materials of the future. They are ideal choices for replacement of metals in the aerospace, automobile, nuclear, and other manufacturing industry applications. Chemical vapor infiltration (CVI) is among the most commonly used techniques for manufacturing composite materials. The CVI process consists of diffusion of reactant gases into a porous preform. Reactants undergo chemical reactions on the surface of the pores to deposit solid material, thereby filling the pores. A detailed two dimensional, isothermal chemical vapor infiltration model is developed in this work.

The basic idea of the numerical computations mentioned above can be described by the development of this model. Also, a physically based model can provide an understanding of the process by providing insight into the transport and reaction mechanisms. It can be used to study the effects of parameter values on reactor performance at a low cost relative to experimental studies. The model can be used in reactor design for specific applications, process scale-up, and for process control applications.

Chapter 2

Framework of the Numerical Computations and Functions

2.1 The MWRtools Functions

MWRtools is a set of Matlab-5 based functions developed for solving boundary-value problems using globally defined trial function expansions and weighted residual methods (MWR). The numerical techniques form a computational toolbox consisting of a common set of numerical tools for implementing the different MWR techniques used in the numerical solution and analysis of systems described by ordinary and partial differential equation models. This library represents the results of efforts to create computational modules that have a one-to-one correspondence with each step of implementing eigenfunction expansion, Galerkin projection, collocation, and other weighted residual methods. Some of the basic MWRtools functions directly related to the data structures described in this work are described briefly below. More information can be found in [1, 2, 13].

Problem Setup

$[\hat{\mathbf{x}}, \hat{\mathbf{w}}, \hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{Q}}] =$ `pd`('geom', M)

Sets up the physical domain in terms of discrete points in the unit interval and defines the differentiation operators according to any imposed problem symmetries. This function is normally called first in the solution procedure. The inputs are the geometry ('slab', 'disk', 'sphe', or 'peri') and the number of discretization points M. The output consists of the discretization grid $\hat{\mathbf{x}}$, quadrature weights $\hat{\mathbf{w}}$, the first-order differentiation array $\hat{\mathbf{A}}$, and the discrete equivalent to the Laplacian operator $\hat{\mathbf{B}}$. This function also computes the discretized set of M Jacobi polynomials $\hat{\mathbf{Q}}$, a discrete transformation array used for filtering and spectral decomposition applications. $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are used in the **loper** object, $\hat{\mathbf{w}}$ is used in the **tfun** object.

$[\boldsymbol{\Psi}] =$ `gdf`($\hat{\mathbf{x}}$, 'f($\hat{x}$,p)', '$\hat{x}$', {'p', $\mathbf{p}$})

Generates discretized representations of a sequence of trial functions $\boldsymbol{\Psi}$ according to the alphanumeric input formula f($\hat{x}$,p), the fine-scale discretization grid $\hat{\mathbf{x}}$ and the parameter p. The cell array contents consist of the parameter name 'p' and its numerical value(s) $\mathbf{p}$. Additional parameters can be specified by concatenating the parameter name/value pairs. The functions can be polynomials, eigenfunctions generated by the explicit solution to a Sturm-Liouville problem, or an arbitrary sequence of functions chosen as part of a Galerkin discretization. This function is normally called after `pd.m`. $\boldsymbol{\Psi}$ is used in the **tfun** object developed in this thesis.

$[\boldsymbol{\lambda}, \boldsymbol{\Psi}, \boldsymbol{\Phi}, \mathbf{w_{ef}}, \mathbf{w_{ad}}] =$ `sl`('geom', $\hat{\mathbf{A}}$, $\hat{\mathbf{x}}$, a, b, c, d, $\hat{\mathbf{w}}$, $\hat{\mathbf{v}}$, $\hat{\mathbf{p}}$, $\hat{\mathbf{q}}$, $\hat{\mathbf{g}}$, $errtol$, a_1, b_1, c_0, d_0)

A Sturm-Liouville problem solver, that computes the vector of eigenvalues $\boldsymbol{\lambda}$, discretized, orthonormal eigenfunctions $\boldsymbol{\Psi}$, adjoint eigenfunctions $\boldsymbol{\Phi}$, discretized weight function $\mathbf{w_{ef}}$ used to define the orthogonality of the eigenfunctions, and discretized weight function $\mathbf{w_{ad}}$ corresponding to the adjoint eigenfunctions. Input includes the problem geometry factor "geom" (geom='slab' or 'peri' corresponds to $\alpha = 0$; geom='disk' to $\alpha = 1$; and geom='sphe' to $\alpha = 2$), and the differentiation, discretization point, and quadrature arrays, $\hat{\mathbf{A}}$, $\hat{\mathbf{x}}$, and $\hat{\mathbf{w}}$, respectively. This function is normally called after `pd.m`. The maximum value *errtol* of the infinity-norm bound is used to discard inaccurate eigenfunctions; a negative value disables error control. $\boldsymbol{\Psi}$ is used in the **tfun** object developed in this thesis.

Operational Functions

$[\mathbf{I_p}] = \texttt{wip}(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{w}})$

The weighted inner product of two sets of discretized functions $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$. This is one of the most heavily used subprograms since inner products are used in nearly every MWR step. The array of inner products $\mathbf{I_p}$ is computed as the vector dot product of the quadrature weight array $\hat{\mathbf{w}}$ with the term-by-term (Hadamard-Schur) product of all combinations of the discretized functions represented in $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$. The number of rows in $\mathbf{I_p}$ corresponds to the number of trial functions in $\hat{\mathbf{G}}$; the columns correspond to the functions in $\hat{\mathbf{F}}$.

Note that this function is overloaded in the object-oriented framework developed in this thesis.

Solution Reconstruction and Refinement

$[\mathbf{T}] = \texttt{sp2pd}(\mathbf{A}, \boldsymbol{\psi}, \boldsymbol{\phi}, ..., \boldsymbol{\eta})$

An important operation is the conversion of the spectrally discretized solution to a physical-space description. It is the spatially discretized equivalent to

$$T(x_1, x_2, ..., x_p) = \sum_{l_1, l_2, ..., l_p = 1}^{L_1, L_2, ..., L_p} a_{l_1, l_2, ..., l_p} \phi_{l_1}(x_1) \psi_{l_2}(x_2) \cdots \eta_{l_p}(x_p)$$

where the input arguments take the form $\mathbf{A}^{L_1 \times L_2 \times ... \times L_p}$, $\boldsymbol{\psi}^{M_1 \times L_1}$, $\boldsymbol{\phi}^{M_2 \times L_2}$, and $\boldsymbol{\eta}^{M_p \times L_p}$. The operation takes place as a sequence of matrix multiplications and permutations, described by

$$\begin{aligned}
\mathbf{T}_1^{M_1 \times L_2 \times ... \times L_p} &= \boldsymbol{\psi}^{M_1 \times L_1} \mathbf{A}^{L_1 \times L_2 \times ... \times L_p} \\
\bar{\mathbf{T}}_1^{L_2 \times M_1 \times ... \times L_p} &= \text{permute}(\mathbf{T}_1^{M_1 \times L_2 \times ... \times L_p}) \\
\mathbf{T}_2^{M_2 \times M_1 \times L_3 \times ... \times L_p} &= \boldsymbol{\phi}^{M_2 \times L_2} \bar{\mathbf{T}}_1^{L_2 \times M_1 \times L_3 \times ... \times L_p} \\
\bar{\mathbf{T}}_2^{L_3 \times M_2 \times M_1 \times ... \times L_p} &= \text{permute}(\mathbf{T}_2^{M_2 \times M_1 \times L_3 \times ... \times L_p}) \\
&\vdots \\
\mathbf{T}_p^{M_p \times M_{p-1} \times ... \times M_1} &= \boldsymbol{\eta}^{M_p \times L_p} \bar{\mathbf{T}}_{p-1}^{L_p \times M_{p-1} \times ... \times M_1} \\
\mathbf{T}^{M_1 \times M_2 \times ... \times M_p} &= \text{permute}(\mathbf{T}_p^{M_p \times M_{p-1} \times ... \times M_1})
\end{aligned}$$

2.2 Functions for High Dimensional Computation

Setting up a higher-dimensional computing system motivates the development of the generalization of matrix multiplication to higher dimensions, where the algebra involved in the generalized matrix multiplication operations is as follows:

$$\begin{aligned}\mathbf{A}^{(L_1\times L_2\times\ldots\times L_p)\times(M_1\times M_2\times\ldots\times M_q)} &= \mathbf{A}^{\mathbf{L}\times\mathbf{M}}\\ \mathbf{B}^{(M_1\times M_2\times\ldots\times M_q)\times(N_1\times N_2\times\ldots\times N_r)} &= \mathbf{B}^{\mathbf{M}\times\mathbf{N}}\end{aligned}$$

implies the generalized matrix multiplication operation

$$\mathbf{A}^{\mathbf{L}\times\mathbf{M}}\mathbf{B}^{\mathbf{M}\times\mathbf{N}} = \mathbf{C}^{\mathbf{L}\times\mathbf{N}}$$

where

$$C_{l_1,l_2,\ldots,l_p,n_1,n_2,\ldots,n_r} = \sum_{k_1=1,\ldots,k_q=1}^{M_1,\ldots,M_q} A_{l_1,l_2,\ldots,l_p,k_1,k_2,\ldots,k_q} B_{k_1,k_2,\ldots,k_q,n_1,n_2,\ldots,n_r}$$

Given this definition, we can define the generalized transpose operation as

$$\left[\mathbf{A}^{\mathbf{L}\times\mathbf{M}}\right]^{\mathrm{T}} = \mathbf{A}^{\mathbf{M}\times\mathbf{L}}$$

a square array as

$$\mathbf{A}^{\mathbf{L}\times\mathbf{M}} \quad \text{such that} \quad L_1 = M_1, L_2 = M_2, \ldots L_q = M_q, \text{ and } p = q$$

the identity array as a square array **A** with all zero elements except

$$\mathbf{I}^{\mathbf{L}\times\mathbf{M}}: \quad I_{l_1,l_2,\ldots,l_q,l_1,l_2,\ldots,l_q} = 1$$

and the matrix inverse as

$$\left[\mathbf{A}^{\mathbf{L}\times\mathbf{M}}\right]^{-1}\mathbf{A}^{\mathbf{L}\times\mathbf{M}} = \mathbf{I}^{\mathbf{L}\times\mathbf{M}}$$

Some functions can perform computations for high-dimensional systems are listed in Figure 2.2 and described below

$[\mathbf{C}] =$ `mprod`$(\mathbf{A}, \mathbf{B}, p, q, r)$

Computes the generalize matrix multiplication by reshaping the arrays **A** and **B** to 2-dimensional arrays and then performing a standard multiplication. **A** is a $p+r$ dimensional array and **B** is a $q+r$ dimensional array. The result is reshaped to the appropriate $p+r$ dimensional array. The arrays **A** and **B** are padded with singleton dimensions for the cases $p = 0$ and $r = 0$; $q = 0$ is equivalent to the Kronecker tensor product of **A** and **B**, reshaped to a $p + r$ dimensional array .

[**x**] = `msolve`(**A**, **b**)

Computes the solution **x** to **Ax** = **b**, when **A** and **b** are not necessarily 1 or 2-dimensional arrays. In its current form, array **A** must be square according to the definition of a square n-dimensional array discussed in the previous section. If **A** is a $p+p$ dimensional array and **b** is a **p** dimensional array, then the solution array **x** is **p** dimensional. The idea here is that **A** will be reshaped to a 2-dimensional array and **b** will be reshaped to a vector. Then the matrix division $\mathbf{A} \setminus \mathbf{b}$ will be implemented to create a solution vector **x**. Finally, **x** will be reshaped to have the same size as **b**.

[**B**] = `mdiag`(**A**)

Convert an n-dimensional array to a diagonal $n + n$ (square) array.

[**B**] = `extract`(**A**, n)

Extract the n'th element from the last dimension of an arbitrary-dimension array or a cell whose elements have the same number of elements in their highest dimension. For example, if **A** is a array with size $I \times J \times M$, the computation $B = extract(A, 2)$ will create a array **B** which is equal to **A**(:,:,2). **B** has the size $I \times J$.

[**J**] = `makejacobian`(**T**, **Jc**)

Set up the Jacobian array in its proper format. **Jc** is the cell array storing the derivatives of the **T** with respect to the state variables. `makejacobian.m` will arrange the **Jc** to a correct format. For more details, please check section 4.6.2.

[**y**] = `moper`(**A**, **f**, *dir*)

Apply a linear operator to a multidimensional array along a specific direction *dir* on the physical domain. The dimensions of the multi-dimensional array A will be rearranged so that they are in the order specified by the vector $[direc, 1 : (direc-1), (direc+1) : ndims(f)]$. Then the multi-dimensional matrix multiplication will be implemented between A and f. Finally, the dimensions of the solution array are rearranged so that y has the same order of the subscripts of A.

[**C**] = `msum`(**A**, **B**)

Compute term-by-term sums of elements in two double/cell arrays. A and B must have the identical structures. If **A** and **B** are double arrays, $\mathbf{C} = \mathbf{A} + \mathbf{B}$. If **A** and **B** are cell arrays, $\mathbf{C}\{i,1\} = \mathbf{A}\{i,1\} + \mathbf{B}\{i,1\}$.

[**tout**,**Y**,**Q**,**tfine**,**phi**,**dYdp**] = `odaepc`(fn,**tint**,**y0**,**param**,**ffun**,**C**,**T**, M)

Polynomial collocation-based ODE/AE solver integrating a system of differential equations $y' = F(t, y)$ from time T0 to TFINAL with initial conditions **y0**. `tint` = [`T0 TFINAL`]. fn is a string containing the name of an ODE file which contains the information of residual functions and Jacobian array. **C** is the capacitance cell array and M is the number of collocation points in time. **Y** is the solution cell array. Time is returned in column vector **tout**.

2.3 Object-Oriented Programming

2.3.1 Introduction

The need for Object-Oriented software is motived by the economic benefits of software reuse, and the opportunity it affords for improvements in the design and software maintenance of large software systems. Software users are looking for programs that are easy to use and, of course, produce correct results in the shortest possible time. Software developers, on the other hand, want computer programs that are easy to understand, maintain, and extend. Whenever possible, new software systems should be assembled from previous developed software components and frameworks. Software modules should be designed so that they can be easily adapted to the changing requirements of ongoing development and employed for multiple projects.

Rather than looking at the actions an application will perform, an Object-Oriented problem solution focuses on the problem data and the operations that can manipulate the data. Object-oriented technology provides mechanisms for explicitly dealing with information hiding through encapsulation, inheritance, and polymorphism enable systems of objects and their behavior to be represented in a manner that is efficient and amenable to software reuse. In MATLAB, Object-Oriented programming is the process of translating the computer-world objects into source code called "classes".

2.3.2 Encapsulation

Encapsulation is the process of bundling related data and functions into wrapped objects. It is the key to Object-Oriented program design. The data and functions may belong together because they have similar properties, or perhaps common behavior. For example, trial functions, their weighting functions and corresponding physical domain directions could be encapsulated in one class. In Object-Oriented terminology, and particularly in MATLAB, the wrapper object is called a class, the functions inside the class are called methods.

2.3.3 Polymorphism

Polymorphism is the ability of an object or object operation to assume different forms. Polymorphism is an important tenet of Object-Oriented programming because it enhances the flexibility of methods and classes within a class hierarchy. In the case of MATLAB, polymorphism is the ability of methods and operators to have meanings that depend on the context in which they are used. Polymorphism comes in the flavor of "overloading". MATLAB enables the programmer to overload most operators to be sensitive to the context in which they are used. Some operators are overloaded frequently, especially the assignment operators and various arithmetic operators such as * and +. The job performed by overloaded operators also can be performed by explicit function calls, but operator notation is usually easier to read.

In classical programming languages, separate tasks are nearly always implemented with separate functions. But sometimes situations arise where two tasks are very similar and perhaps only differ in the function arguments they use. The programmer wants give them the same function name and let the computer use the correct one based on the type of the arguments you give. That is exactly what overloading is about.

Suppose, for example, that we want to define a matrix multiplication method.

This method can work for a **loper** class and a double array. It also can do the multiplication between a **loper** class and a **tfun** class. In MATLAB, we can define two methods with separate source codes but with the same name :

```
mtimes(loper, double);
mtimes(loper, tfun);
```

At run-time, if we call the method "mtimes" with **loper** and double arguments, the first method will be called. If we call the function with **loper** and **tfun** as arguments, then the second method will be called.

2.4 The Classes and Their Methods

When using well-designed classes, Object-Oriented programming can significantly increase code reuse and make the programs easier to maintain and extend. Because object properties are not visible from the command line, they can only be accessed with class methods. This protects the object properties from operations not intended for the object's class. Also, we can create methods that override existing MATLAB functions.

Currently, three classes were written on the basic MWR computational functions described above. A discussion of the classes with their methods is given below.

loper

[L] = `loper`(v,dir)

This is the constructor method of the **loper** class. It is a special method that initializes the data members of a class object. A class's constructor method is called automatically when an object of that class is created. The constructor has the same name as the class. An object named L belonging to the class **loper** is created when this method is called. v is a two dimensional first-order differentiation or discrete equivalent to the Laplacian array generated by pd.m. dir is a scalar denoting the direction in physical space.

[c] = `mtimes`(L,B)

Computes $c = LB$. This method overloads the matrix multiplication operator "$\times$". L and B can be a **loper** object and a double array, respectively. B can also be a **tfun** object. A double array c will be created.

[p, q] = `get`(L)

This method can get and return the properties of a **loper** class object L. p is the discretized differentiation array and q is its direction in physical space. Properties of a **loper** object can not be accessed from outside of the **loper** class. When properties of a **loper** object are needed, a **loper** class method such as **get** can be called.

tfun

A truncated trial function sequences class that contains all the sets of trial functions used to discretize a distributed state variable. The direction of each sequence and the weighting function of each trial function is indicated in the definition of this object.

[F] = `tfun`(t, dir, w)

This is the constructor method of the **tfun** class. An object of the **tfun** class is created when this method is called. All sets of the trial functions are included in the cell array t. dir is a vector containing the directions in physical space. And w is a cell array containing all the sets of quadrature weight arrays.

[p, q, w] = `get`(F)

This method can get and return the properties of a **tfun** class object F. The cell array p contains the trial functions, the vector q contains the directions and the cell array w contains the weighting functions.

[C] = `tfunset`(F,L)

Computes $C = FL$, where F is a **tfun** object and L is a **loper** object. The property v of L multiplies a trial function set t in F where t has the same direction as L. A **tfun** object C is created. C has the same weights and directions as F.

[c] = `mtimes`(a,F)

Computes $c = aF$, this method overloads the matrix multiplication operator $\times$. a is a double array, F is a **tfun** object, and c is a double array. The MWR function $c = sp2pd(a, F.t1, F.t2...)$ is called inside this method.

[S] = `times`(a,F)

Computes $S = a. * F$. The term-by-term multiplication operator "$.\times$" is overloaded using this method. a is a double array and F is a **tfun** object. This method creates an **sfield** object S that has the same directions as F.

[c] = `wip`(A,F)

This method performs the quadrature-based inner product computation. The **wip** function in the MWRtools is overloaded here. A is a double array or a tfun object, F is a tfun object, producing a double array c.

sfield

A scalar field object class which is used for intermediate computations.

[S] = `sfield`(A,dir)

This is the constructor method of the **sfield** class. The **sfield** class has two properties: field and dir. The input argument A can be a cell array of trial functions and the input argument dir contains the directions. If A is a object of the tfun class, the properties val and dir in this **tfun** object can be taken as the properties of S. If a double array multiplies a **tfun** object term-by-term, a **sfield** object will be created which has the same dir property as in the **tfun**.

[p,q] = `get`(S)

This method can get and return the properties of a **sfield** object S. Cell array p contains the property field of S and q contains the property dir of S.

[c] = `wip`(S,F)

This method overloads the weighted inner product operator. S a is **sfield** object, F is a tfun object. It returns a double array c.

2.5 A Simple Example

We use a relatively simple example here to show the basic ideas described in the previous sections.

Consider a catalytic pellet inside a gas phase CSTR. If $c(t,r,z)$ is the reactant species concentration inside the catalyst pellet, the modeling equations for the process at steady state can be written as

$$0 = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right) + \frac{\partial^2 c}{\partial z^2} - \phi^2 c^2 \tag{2.1}$$

subject to boundary conditions

$$\frac{\partial c(t,0,z)}{\partial r} = 0 \qquad c(t,1,z) = 1 \qquad c(t,r,1) = 1 \qquad \frac{\partial c(t,r,0)}{\partial z} = 0$$

Here ϕ is the Thiele modulus. The catalyst pellet-phase concentration profile is expressed in terms of the truncated trial function expansion

$$c_{I,J}(r,z) = 1 + \sum_{i,j=1}^{I,J} a_{i,j}(t)\eta_i(r)\psi_j(z) \tag{2.2}$$

where η_i and ψ_j are computed as the eigenfunctions satisfying $\lambda^r\eta = \nabla_r^2\eta$ subject to $\eta'(0) = 0$, $\eta(1) = 0$ and $\lambda^z\psi = \nabla_z^2\psi$ subject to $\psi'(0) = 0$, $\psi(1) = 0$, respectively.

The physical domains can be set up with the MWRtools functions :

```
[r,wr,dr,ddr] = pd('disk',40) ;

[z,wz,dz,ddz] = pd('slab',40) ;
```

The eigenfunctions can be computed with the MWRtools library functions :

```
[lam_r, eta] = sl('disk',dr,r,1,0,0,1,wr) ;

[lam_z, psi] = sl('slab',dz,z,1,0,0,1,wz) ;
```

Then we create a **tfun** object `P` and two **loper** objects `DDR` and `DDZ` :

`P = tfun({eta, psi}, [1  2], {wr, wz});`

`DDR = loper(ddr, 1);`

`DDZ = loper(ddz, 2);`

The Galerkin projection solution is implemented with the following steps :

The residual function rhs is computed by substituting equation (2.2) into equation (2.1) and is projected on each trial function to generate the $I \times J$ array **rhs**:

$$rhs = \langle R, \eta\psi \rangle = 0$$

We use Newton's method to solve this set of nonlinear equations:

$$\mathbf{a}^{\nu+1} = \mathbf{a}^{\nu} - \mathbf{Jac}^{-1}\,\mathbf{R} \tag{2.3}$$

where **Jac** is the $I \times J \times I \times J$ Jacobian array.

```
for iters = 1:8

    c = 1 + a*P ;

    R = DDR*c + DDZ*c - phi^2*c^2 ;

    rhs = wip(R,P) ;

    Jac = wip(DDR*P,P) + wip(DDZ*P,P) - wip(phi^2*2*c.*P,P) ;

    update = msolve(Jac,rhs);

    a = a - update ;

end
```

The solution is reconstructed in the physical space using

`c = 1 + a * P;`

The results with different value of Thiele modulus are plotted in Figure 2.4.

MWR Elements	[Output]	= `Function`	(Required Input,	Optional Input)

Specify geometry
Trial functions
and
eigenfunctions
	$\hat{\mathbf{x}}, \hat{\mathbf{w}}, \hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{Q}}$
	$\boldsymbol{\Psi}$
	λ, $\boldsymbol{\Psi}$, $\boldsymbol{\Phi}$,
	$\mathbf{w_{ef}}$, $\mathbf{w_{ad}}$
		`pd`
		`gdf`
		`sl`
			'geom', M
			$\hat{\mathbf{x}}$, 'f($\hat{x}$,p)', '$\hat{x}$'
			'geom', $\hat{\mathbf{A}}$, $\hat{\mathbf{x}}$, ...
			a, b, c, d, $\hat{\mathbf{w}}$
				
				{'p', $\mathbf{p}$}
				$\hat{\mathbf{v}}$, $\hat{\mathbf{p}}$, $\hat{\mathbf{q}}$, $\hat{\mathbf{g}}$, ϵ, ...
				a_1, b_1, c_0, d_0

Projection	$\mathbf{I_p}$	`wip`	$\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{w}}$	
Solution refinement	$\mathbf{T}$	`sp2pd`	$\mathbf{A}$, ϕ, $\mathbf{psi}$, $\ldots$, η	

Figure 2.1: The subset of current MWRtools functions, with function output, function name, required input, and optional input.

MWR Elements	[Output]	= Function	(Input)
Array multiplication	**C**	`mprod`	**A**, **B**, p, q, r
Solve $\mathbf{Ax} = \mathbf{B}$	**x**	`msolve`	**A**, b
Diagonal array	**B**	`mdiag`	**A**
Extract elements	**B**	`extract`	**A** , n
Jacobian array	**J**	`makejacobian`	**T** ,**Jc**
operator multiplication	**y**	`moper`	**A**,**f**,dir
Term-by-term sums	**C**	`msum`	**A**,**B**
ODE/AE solver	[**tout**,**Y**,**Q**, **tfine**,**phi**,**dYdp**]	`odaepc`	fn, **tint**,**y0**, **param**, **ffun**,**C**,**T**,M

Figure 2.2: The table of high dimensional MWR functions, with function output, name, function input.

Classes	*Methods*	[Output]	= `Method`	(Input)
Loper	constructor	L	`loper`	v, dir
	get propertities	p, q	`get`	L
	$L \times B$	c	`mtimes`	L , B
Tfun	constructor	F	`tfun`	t, dir, w
	get propertities	p, q, w	`get`	F
	tfun times loper	C	`tfunset`	F , L
	$a \times F$	c	`mtimes`	a , F
	a. $\times$ F	S	`times`	a , F
	inner product	c	`wip`	A , F
Sfield	constructor	S	`sfield`	A, dir
	get properties	p, q	`get`	S
	inner product	c	`wip`	S, F

Figure 2.3: The table of classes and their methods.

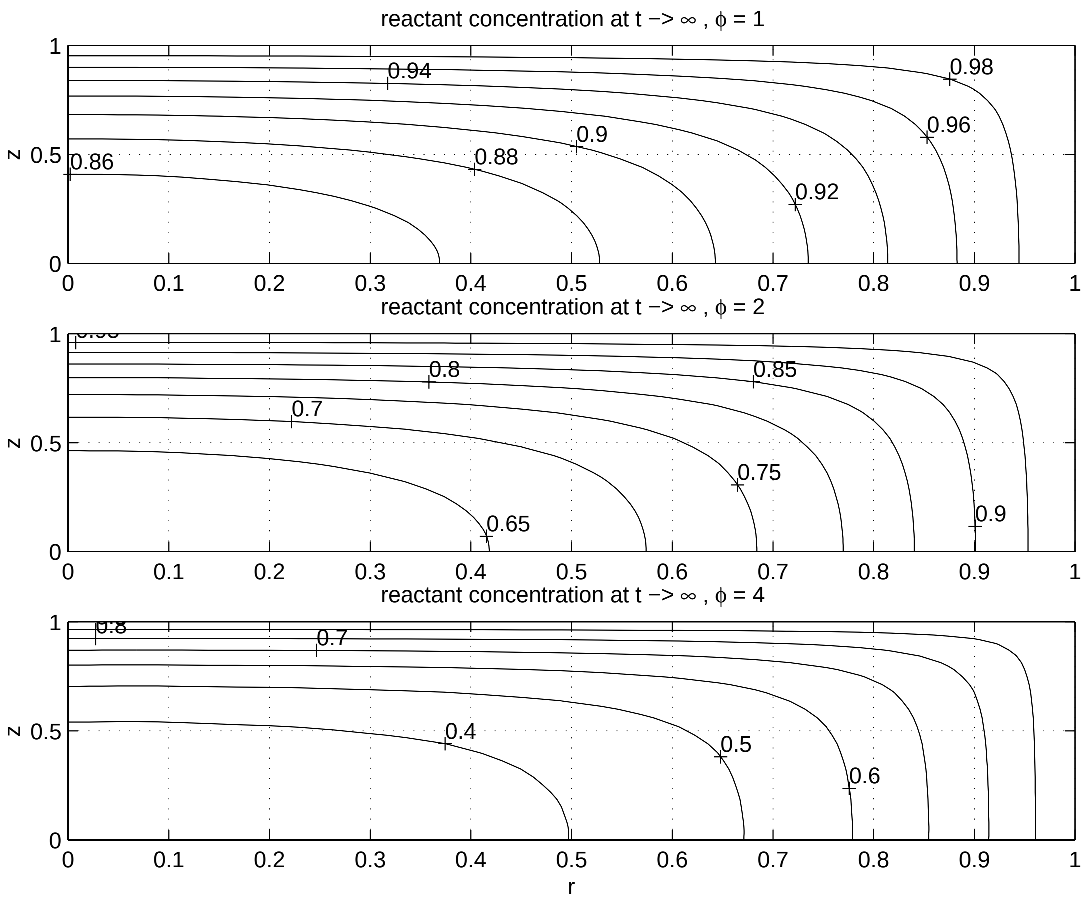

Figure 2.4: Reactant species concentration in a catalytic pellet at steady state. Trial function truncation number along r direction is set at 14. Trial function truncation number along z direction is set at 16.

Chapter 3

CVI Process Modeling

3.1 Introduction

3.1.1 Carbon Fiber - Carbon Matrix Composite Materials

Composites are defined as materials in which two or more constituents have been brought together to produce a new material, which has properties different from those of the individual constituents. One of the most critical aspects determining the nature and properties of a composite material is the interfaces between components. The strength of the composite is often governed by the adhesion forces which can be physical, chemical or a combination of the two [5, 23].

Carbon fiber-carbon matrix composites form a very specialized group of composite materials. This kind of composite consists of carbon fibers embedded in a carbon matrix (see Figure 3.1). The aim of these materials is to combine the advantages of fiber-reinforced composites such as high specific strength, stiffness and in-plane toughness with the refractory properties of structural ceramics. The microstructure will differ according to the type of raw materials and the processing conditions. The differences in microstructure exert a considerable influence on the properties of the materials.

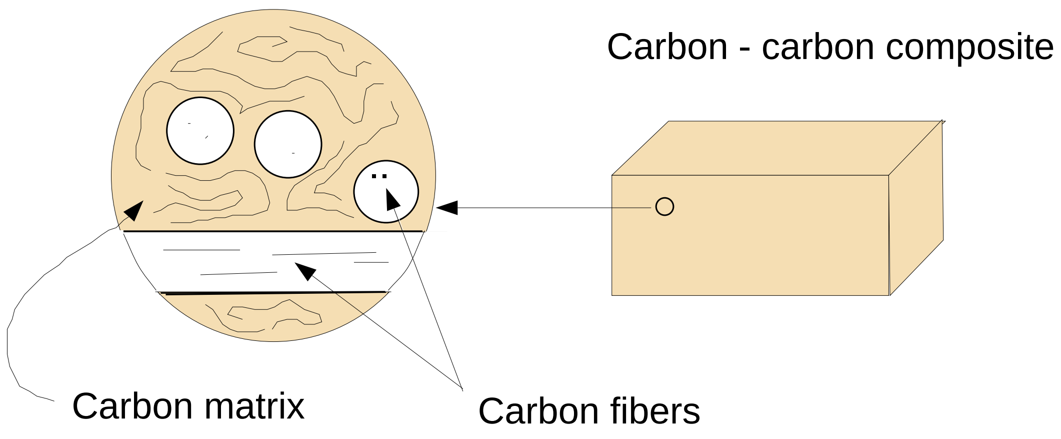

Figure 3.1: Closeup of a carbon-carbon composite

Carbon fibers are filaments consisting of non-graphite carbon produced by the carbonization of synthetic or natural organic fibers or of fibers spun from organic precursors such as resins and pitches [5]. In the majority of cases (fabricated using not very high heat treatment i.e., temperature less than 2700° C), the fibers remain as non-graphite carbon.

Reinforced carbon fiber - carbon matrix composites are gaining technical importance in aerospace, automobile and manufacturing industries because their unique characteristics: they retain strength at high temperature and weigh far less than their metal equivalents. Over 60% by volume of the carbon-carbon produced in the world is used in braking systems, especially for aircraft [5, 23]. It is also used to make rocket nozzles, and components of racing cars, high-speed trains and main battle tanks. Furthermore, because of its extremely good biocompatibility, it is of interest as a replacement for metals used in implant surgery.

3.1.2 Chemical Vapor Infiltration (CVI)

The chemical vapor infiltration (CVI) approach is one reaction-forming processes for making these carbon fiber-carbon matrix composites. In CVI processes, reactant gases are introduced to a hot porous preform where they react on the surface of the fibers to form a solid coating. The coating grows with continued deposition to form the matrix. As the deposits become thicker, they begin to impinge and grow into one another to form a continuous solid matrix between the fibers. The porosity decreases until pore channels become plugged or fully densified. The main advantage of this process is that relatively low process temperature can be used to deposit the matrix without damaging the fibers. Also, there is the potential for many different matrix materials because of the variety of gas-solid reaction chemistries available.

There have been several types of CVI processes proposed during the years. they are generally classified as isothermal, thermal gradient, isothermal-forced flow, thermal gradient-forced flow, and pulse flow.

The primary objectives of CVI are to maximize the rate of matrix deposition and minimize density gradients. Unfortunately, there is an inherent competition between the deposition reaction and the mass transport of the gas reactants into the preform. The deposition occurs preferentially near the outer surface of the preform because the concentration of the reactant is at its maximum value near the outer surface; the concentration decreases significantly due to consumption of the reactant inside the preform. As a result, when the process is carried out isothermally surface pores tend to close first, restricting the gas transport to the interior of the preform. This phenomenon is called 'canning' [24].

The isothermal CVI process must be carried out at low temperatures and pressures to enhance infiltration. Chemical reaction rather than diffusion should be the rate-limiting step in the deposition process. Diffusion is a slow process, and the low temperature and pressure lead to very low deposition rates. Hence, this method requires infiltration times on the order of several weeks and is restricted to thin components. These limitations add considerably to the cost of components and limit the application of this material.

3.2 CVI Reactor Systems

Mathematical models can provide insight into the physicochemical processes governing CVI as well as valuable guidelines for experimental research. The use of mathematical models can avoid time consuming and expensive trial and error practices and help in rapid evaluation of novel reactor designs and modes of operation.

Modeling studies of CVI have been increasing in number and complexity in the last several years. There is a large literature on the mathematical modeling and simulation of different CVI systems. McAllister and Wolf [16] developed a model for isothermal CVI of carbon-carbon composite substrates during propylene pyrolysis. The orthogonal collocation method was used along with a Runge-Kutta technique in their study. The numerical solutions were compared to predict experimental results. Chung and McCoy [6] studied a CVI process at steady state with layered, woven fabrics. Gupte and Tsamopoulos [12] developed a forced flow CVI model for ceramic composites. Skamser and Johnson [24] simulated the deposition of alumina matrix within fibers using thermal gradients to optimize the process parameters. Vaidyaraman and coworkers [27] simulated a forced flow-thermal gradient CVI process for carbon-carbon composites. Their work focused on the reaction kinetics instead of gas transport. However, all of the above are restricted for one dimensional problems. Although one dimensional models are adequate for a preliminary evaluation of the process, high dimensional models are necessary to account for the finite geometry of the preform. Furthermore, important approximations in the physical description of the process introduced by one dimensional models can be eliminated when using high dimensional models. Middleman [18] developed a two dimensional single pore model at steady state. McAllister and Wolf [17] modeled a two dimensional CVI process of a multiple substrate reactor. Deepak and Evans [7] developed a mathematical model for CVI in a microwave-heated preform. A simplified dusty gas model was used in their study. Morell and coworkers [21] modeled a two dimensional CVI process with volume heating. A dusty gas model was used to describe the multi-component gaseous species diffusion and the modeling equations were solved by the method of lines. Ofori and Sotirchos did research on the pore structures inside the preform [25] and they also developed two and three dimensional CVI models [22] with a generalized form of the dusty gas model. The model equations were solved by the Galerkin finite element method. Midha and Economou [19, 20] developed a model of two dimensional CVI with radio frequency heating. Isothermal and thermal gradient simulation are compared and the governing equations were discretized spatially using a Galerkin finite element formulation.

A two dimensional simulation of isothermal isobaric CVI is presented in this thesis work because this is the most widely accepted model of CVI. The model is developed based on the work of Midha and Economous [19, 20]. In their work, a dusty-gas model was used to describe the multi-component diffusion mechanism. Because the emphasis of this thesis work is on the numerical techniques rather than the details of the model development, we assume only two gas species methane and hydrogen are involved in the CVI process. Therefore, a simplified gas diffusion mechanism is used in our work.

The analysis of the densification of a porous substrate by CVI requires the consideration of diffusion and reaction of gases in a porous substrate occurring simultaneously with the deposition of a solid product. The work presented here is to incorporate a scheme for the formation of carbon deposits during the pyrolysis of methane into a model which accounts for the diffusion, reaction and porosity changes occurring during carbon infiltration. This model enables the prediction of the porosity as functions of reaction conditions and time of deposition. The porosity is defined as the ratio of void volume to the total volume of the preform. The modeling partial differential equations are generated from reacting species mass balances on CH_4 , H_2, and the deposited carbon matrix (which is directly related to porosity).

The reactor system modeled in this work is shown in Figure 3.2 which is the same as in [20]. The cylindrical carbon perform is placed coaxially in a cylindrical quartz reactor. A radio frequency (RF) induction coil surrounds the reactor and is used to provide volumetric heating of the preform to desired temperature. The gaseous reactant flows continuously in the reactor, maintaining a constant species concentration at the surface of the preform. The gaseous precursor diffuses into the porous preform and chemically reacts at the elevated temperature to deposit solid carbon within the pores.

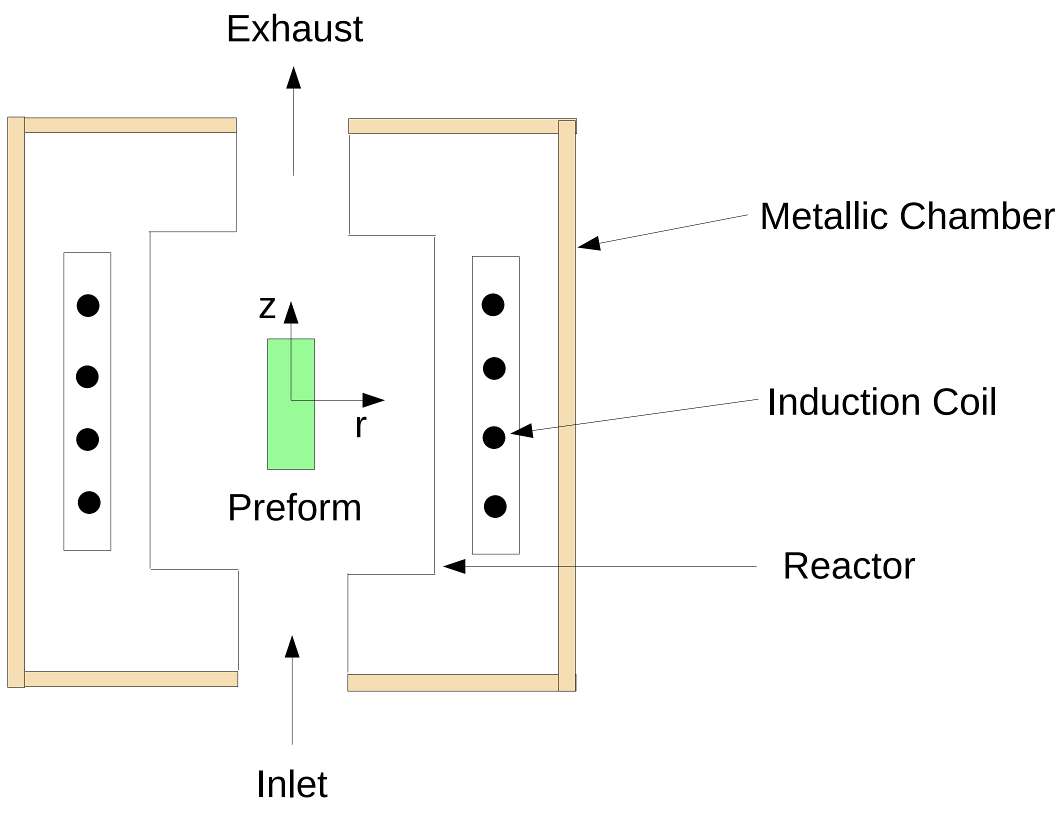

Figure 3.2: The isothermal CVI system used in [20].

3.3 Mathematical Model

Mathematical modeling of the CVI process involves the description of the transport and reaction phenomena occurring inside the composite. The system is characterized by the time evolution of species concentration and pore structure. In general, important processes include the diffusion of gaseous species into and out of the fiber-matrix composite and the chemical reaction.

The system under consideration is a preform of cylindrical geometry with radius R^* and height $2Z^*$. Due to the symmetry considerations, the domain of the CVI model consists only of the upper-right quadrant of the preform (see Figure 3.3). The fibrous structure consists of cylindrical fibers randomly oriented in three dimensional space.

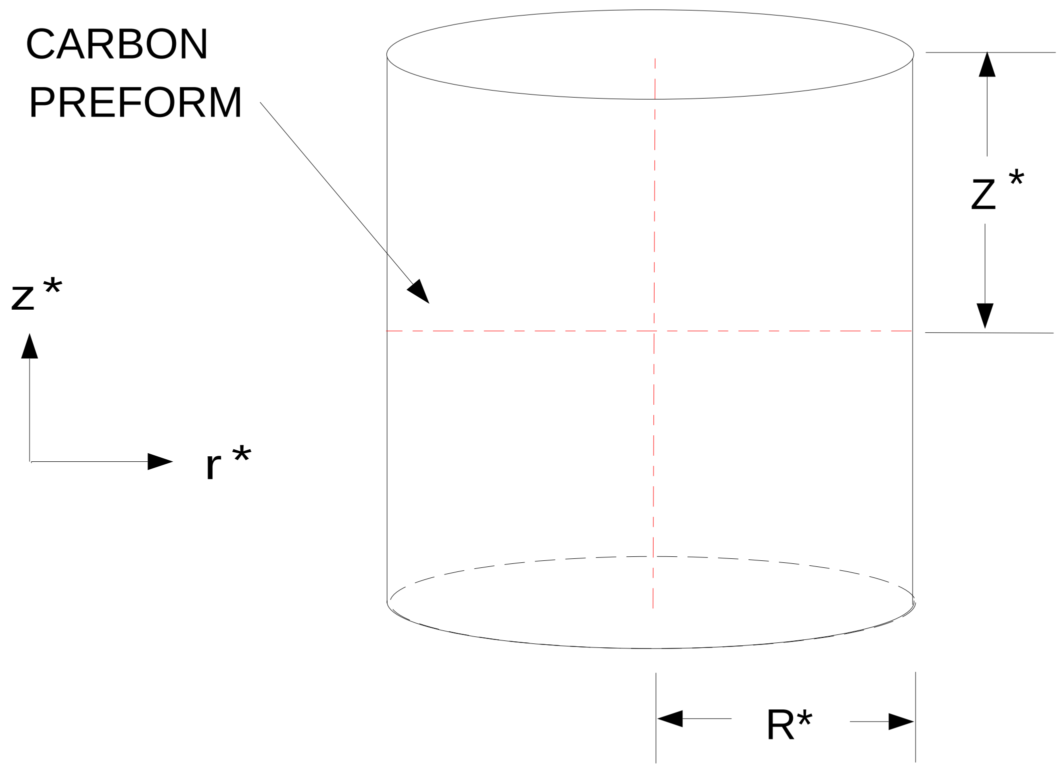

Figure 3.3: Closeup of the carbon preform

The deposition of carbon by the decomposition of methane is taken as a model chemical system in the present work. Despite the fact that the gas phase chemistry is relatively well understood, knowledge of surface processes is still very incomplete. Experimental investigations [21] have shown that the decomposition reaction is first order in methane concentration. Hence, simplified chemical kinetics are used whereby the deposition of carbon is described by an overall reaction in the reactor :

$$CH_4(gas) \rightarrow 2H_2(gas) + C(solid)$$

was assumed with the rate of reaction given by

$$R_{rxn} = C_{CH_4} k\ exp(-\frac{E}{RT}) \tag{3.1}$$

$$R_{CH_4} = -C_{CH_4} k\ exp(-\frac{E}{RT}) \tag{3.2}$$

$$R_{H_2} = 2C_{H_2} k\ exp(-\frac{E}{RT}) \tag{3.3}$$

where C_{CH_4} and C_{H_2} are the concentration of CH_4 and H_2 in the carbon preform, respectively. k is the rate constant and E is the activation energy of the reaction. The values of these parameters were taken as 2.24 $\times 10^{14}$ 1/s and 3.64 $\times 10^{5}$ J/mol, respectively. T is the temperature in the preform.

3.3.1 Equations of Continuity

If we take N_{CH_4} and N_{H_2} to be the molar flux of CH_4 and H_2 , ϵ^e to be the accessible porosity, D_{eff} to be the effective diffusivity of $CH_4 - H_2$, the equation of continuity can be written as :

$$\frac{\partial}{\partial t}(\epsilon^e C_{CH_4}) + \nabla \cdot N_{CH_4} = \epsilon^e R_{CH_4} \tag{3.4}$$

$$\frac{\partial}{\partial t}(\epsilon^e C_{H_2}) + \nabla \cdot N_{H_2} = \epsilon^e R_{H_2} \tag{3.5}$$

where

$$\nabla \cdot N_{CH_4} = \frac{1}{r^*}\frac{\partial}{\partial r^*}(r^* N_{CH_4,r^*}) + \frac{\partial N_{CH_4,z^*}}{\partial z^*} \tag{3.6}$$

$$\nabla \cdot N_{H_2} = \frac{1}{r^*}\frac{\partial}{\partial r^*}(r^* N_{H_2,r^*}) + \frac{\partial N_{H_2,z^*}}{\partial z^*} \tag{3.7}$$

N_{CH_4,r^*} and N_{CH_4,z^*} are the molar flux of CH_4 in r^* and z^* directions. N_{H_2,r^*} and N_{H_2,z^*} are the molar flux of H_2 in r^* and z^* directions. And we can relate the molar flux to the concentration gradient by

$$N_{CH_4,r^*} = -D_{eff}\frac{\partial C_{CH_4}}{\partial r^*} \tag{3.8}$$

$$N_{CH_4,z^*} = -D_{eff}\frac{\partial C_{CH_4}}{\partial z^*} \tag{3.9}$$

$$N_{H_2,r^*} = -D_{eff}\frac{\partial C_{H_2}}{\partial r^*} \tag{3.10}$$

$$N_{H_2,z^*} = -D_{eff}\frac{\partial C_{H_2}}{\partial z^*} \tag{3.11}$$

D_{eff} is the effective binary diffusion coefficient, which is obtained from the following relation [20]

$$D_{eff} = D_{AB}S_1 \tag{3.12}$$

where D_{AB} is the binary diffusion coefficient between CH_4 and H_2 estimated at a reference temperature using the Chapman-Enskog theory [4]:

$$D_{AB} = D_{AB}^{ref}(\frac{P^{ref}}{P})(\frac{T}{T^{ref}})^{1.65} \tag{3.13}$$

in which D_{AB}^{ref} is a reference binary diffusivity at a reference temperature T^{ref} and a reference pressure P^{ref}.

S_1 represents the pore structure parameter which depends on the particular pore structure model used. The pore structure of the three dimensional carbon preform was represented by a network of uniformly sized cylindrical capillaries of a certain radius. S_1 is given by

$$S_1 = \frac{\epsilon^e}{3} \tag{3.14}$$

ϵ is the total porosity of the preform, which is defined as the ratio of void volume of the preform to the total volume of the preform. According to [22], the accessible porosity ϵ^e is equal to the total porosity for values of $\epsilon > 0.1$. Hence, we do not differentiate between the two and we will use symbol ϵ representing porosity in this study.

With these definitions, the equations of mass transport for the gaseous species CH_4 and H_2 become

$$\frac{\partial}{\partial t}(\epsilon C_{CH_4}) = \frac{1}{r^*}\frac{\partial}{\partial r^*}(r^* S_1 D_{AB}\frac{\partial C_{CH_4}}{\partial r^*}) + \frac{\partial}{\partial z^*}(S_1 D_{AB}\frac{\partial C_{CH_4}}{\partial z^*}) + \epsilon R_{CH_4} \tag{3.15}$$

$$\frac{\partial}{\partial t}(\epsilon C_{H_2}) = \frac{1}{r^*}\frac{\partial}{\partial r^*}(r^* S_1 D_{AB}\frac{\partial C_{H_2}}{\partial r^*}) + \frac{\partial}{\partial z^*}(S_1 D_{AB}\frac{\partial C_{H_2}}{\partial z^*}) + \epsilon R_{H_2} \tag{3.16}$$

3.3.2 Overall Mass Balance for the Solid Phase

The evolution of porosity as well as its dependence on position can be obtained from the following equation

$$\frac{d\epsilon}{dt} = -\frac{M_c}{\rho}\epsilon R_{carbon} \tag{3.17}$$

where M_c is the molecular weight of the carbon deposit and ρ is the density of the carbon deposit. Density within the preform is always uniform because we assume the porosity is uniformly distributed throughout the preform. Densities in the range of 1.9 to 2.4 g/cm^3 are typically observed for CVI carbon. Simulation shows that changing the assumed carbon density from 1.8 to 2.3 g/cm^3 changed the value of total porosity very little. The density of carbon 2.27 g/cm^3 in this study is used as the same as in [20]. R_{carbon} is the reaction rate of producing solid carbon:

$$R_{carbon} = C_{CH_4} k \ exp(-\frac{E}{RT}) \tag{3.18}$$

The transport parameters for the structural preform are listed in Figure 3.4.

3.4 Boundary and Initial Conditions

The composition of the mixture in the gas phase surrounding the preform is assumed to be the same everywhere, with negligible mass transport limitations between the gas phase and the external surface of the preform. The composition of the gas mixture at the external surface of the preform is known and equal to that in the feedstock gas. Therefore, at $r^* = R^*$, $z^* = Z^*$,

Parameter	Value	Unit	Description
M_c	12.01	g/mol	molecular weight of carbon
ρ	2.27×10^6	g/m^3	density of carbon
D_{AB}	59.4986	m^2/hr	binary diffusion coefficient
ϵ_0	0.7		initial porosity
$C^R_{CH_4}$	0.685	mol/ m^3	CH_4 concentration out of preform
$C^R_{H_2}$	0	mol/ m^3	H_2 concentration out of preform
T	1400	^oK	temperature in the reactor
P	100	torr	nominal pressure
Z^*	0.0381	m	half of the height of the preform
R^*	0.0142	m	radius of the preform
k	8.064×10^{17}	1/hr	reaction constant
E	3.64×10^5	J/mol	activation energy

Figure 3.4: The table of parameters for the preform from [20]

$$C_{CH_4} = C_{CH_4,R} \tag{3.19}$$

$$C_{H_2} = C_{H_2,R} \tag{3.20}$$

In this study, the feed gas is composed of pure methane. We assume the volumetric flow of CH_4 is sufficient to dilute the H_2 produced to the point where it is negligible. Therefore,

$$C_{H_2,R} = 0 \tag{3.21}$$

We assume that at the center of the preform there is no flux of any of the gaseous species. This gives the boundary conditions at $r^* = 0$ as

$$\frac{\partial C_{CH_4}}{\partial r^*} = 0 \tag{3.22}$$

$$\frac{\partial C_{H_2}}{\partial r^*} = 0 \tag{3.23}$$

Because we are interested in the long-time behavior of the process, the concentration of CH_4 and H_2 is set equal to zero at the initial state. The initial porosity of the preform ϵ_0 was assumed as 0.7 which is uniform through the entire preform.

3.5 Dimensionless Equations

We make the system of equations dimensionless by defining the parameters and variables as in Figure 3.5.

This gives the modeling equations 3.15, 3.16 and 3.17 in dimensionless form as

$$\frac{\partial}{\partial t}(\epsilon C_{CH_4}) = \alpha_r \frac{1}{r}\frac{\partial}{\partial r}(r\epsilon \frac{\partial C_{CH_4}}{\partial r}) + \alpha_z \frac{\partial}{\partial z}(\epsilon \frac{\partial C_{CH_4}}{\partial z}) - \zeta C_{CH_4}\epsilon \tag{3.24}$$

$$\frac{\partial}{\partial t}(\epsilon C_{H_2}) = \alpha_r \frac{1}{r}\frac{\partial}{\partial r}(r\epsilon \frac{\partial C_{H_2}}{\partial r}) + \alpha_z \frac{\partial}{\partial z}(\epsilon \frac{\partial C_{H_2}}{\partial z}) + 2\zeta C_{CH_4}\epsilon \tag{3.25}$$

$$\frac{d\epsilon}{dt} = -\gamma\zeta C_{CH_4}\epsilon \tag{3.26}$$

For equation 3.24, the left hand side can be written as

Dimensionless variables and parameters
$r = \frac{r^*}{R^*}$
$z = \frac{z^*}{Z^*}$
$\alpha_r = \frac{D_{AB}}{3R^{*2}}$
$\alpha_z = \frac{D_{AB}}{3Z^{*2}}$
$\gamma = \frac{M_c}{\rho}$
$\zeta = k\ exp(-\frac{E}{RT})$

Figure 3.5: The table of dimensionless variables and parameters

$$\frac{\partial}{\partial t}(\epsilon C_{CH_4}) = \epsilon \frac{\partial C_{CH_4}}{\partial t} + C_{CH_4}\frac{d\epsilon}{dt}$$

To avoid the singular problem at $r = 0$, the first term in the right hand side of Equation 3.24 is expanded:

$$\begin{aligned} \frac{1}{r}\frac{\partial}{\partial r}(r\epsilon\frac{\partial C_{CH_4}}{\partial r}) &= \frac{\epsilon}{r}\frac{\partial C_{CH_4}}{\partial r} + \frac{\partial \epsilon}{\partial r}\frac{\partial C_{CH_4}}{\partial r} + \epsilon\frac{\partial^2 C_{CH_4}}{\partial r^2} \\ &= \frac{\epsilon}{r}\frac{\partial}{\partial r}(r\frac{\partial C_{CH_4}}{\partial r}) + \frac{\partial \epsilon}{\partial r}\frac{\partial C_{CH_4}}{\partial r} \end{aligned}$$

Then we can rewrite equation 3.24 as the following form

$$\begin{aligned}\frac{\partial C_{CH_4}}{\partial t} &= \frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial C_{CH_4}}{\partial r}) + \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial C_{CH_4}}{\partial r} + \alpha_z\frac{\partial^2 C_{CH_4}}{\partial z^2} + \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial C_{CH_4}}{\partial z} \\ &- \zeta C_{CH_4} + \gamma\zeta C_{CH_4}^2 \qquad (3.27)\end{aligned}$$

We can rewrite equation 3.25 following a similar calculation procedure

$$\begin{aligned}\frac{\partial C_{H_2}}{\partial t} &= \frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial C_{H_2}}{\partial r}) + \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial C_{H_2}}{\partial r} + \alpha_z\frac{\partial^2 C_{H_2}}{\partial z^2} + \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial C_{H_2}}{\partial z} \\ &+ 2\zeta C_{CH_4} + \gamma\zeta C_{CH_4}C_{H_2} \qquad (3.28)\end{aligned}$$

The dimensionless boundary and initial conditions are :

r=0	$\frac{\partial C_{CH_4}}{\partial r} = 0$	$\frac{\partial C_{H_2}}{\partial r} = 0$
r=1	$C_{CH_4} = C_{CH_4,R}$	$C_{H_2} = C_{H_2,R}$
z=0	$\frac{\partial C_{CH_4}}{\partial z} = 0$	$\frac{\partial C_{H_2}}{\partial z} = 0$
z=1	$C_{CH_4} = C_{CH_4,R}$	$C_{H_2} = C_{H_2,R}$
t=0	$C_{CH_4} = 0$	$C_{H_2} = 0$

Chapter 4

Numerical Solution Methods

The Galerkin technique can be seen as an extension of the eigenfunction expansion method to nonlinear systems. The Galerkin method is based on choosing a trial function expansion and projecting the residual onto each function, making the residual orthogonal to the sequence of trial functions. In this work, the Galerkin method is used to spatially discretize the partial differential equations governing the CVI process (described in chapter 3) to a set a ordinary differential equation. The resulting ODE set is integrated by a orthogonal-collocation based time integrator. Our goal is to develop a set of numerical tools for implementing the MWR techniques for high dimensional computation inside the MATLAB computational environment. We introduced Object-Oriented techniques into the development of the numerical tools in the chapter 2 to make the implementing/programming simple and straightforward.

4.1 Expansion of Solution

The concentration of CH_4 and H_2 are expressed in terms of the trial function expansions:

$$C_{CH_4}(r,z,t) = C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z) \tag{4.1}$$

$$C_{H_2}(r,z,t) = \sum_{i=1}^{I}\sum_{j=1}^{J} b_{ij}(t)\phi_i(r)\psi_j(z) \tag{4.2}$$

The porosity value is a function of r and z although its governing equation has a form of an ordinary differential equation. This is because the rate of change of ϵ depends on the concentration of methane C_{CH_4}. Therefore, we must expand the porosity in the form:

$$\epsilon(r,z,t) = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}(t)\eta_i(r)\xi_j(z) \tag{4.3}$$

In these truncated trial function expansions, a, b and c are the mode amplitude coefficients and ϕ, ψ, η, ξ are the trial functions. The means by which the trial functions are represented and computed is described in the following section.

4.2 Fine-scale Discretization

The MWRtools function `pd.m` is used to define the discretized physical domain and the differentiation and quadrature operators according to the specific domain geometry. The method of choosing the number of fine-scale discretization points depends strongly on the problem and aspects of the MWR solution procedure. In general, the number must be greater than the trial function truncation number. Furthermore, accurate residual calculations for nonlinear problems require more discretization points. Furthermore, the choice of the point position must balance the interpolation performance against the accuracy of quadrature weights. In `pd.m`, the fine-scale discretization point locations are chosen as the fixed end points as 0 and 1, and the interior points as the roots of the Jacobi polynomial. Thus, the fine-grid discretization point density increase towards each end of the interval. To demonstrate the computation procedure, we use 14 points on each (r and z) direction. The MATLAB commands are :

```
[r, wr, dr, ddr] = pd('cyln', 14);
[z, wz, dz, ddz] = pd('slab', 14);
```

The distribution of the discretization points on r and z directions is shown on Figure 4.1.

In this work, we denote the r direction is direction 1 and z direction is direction

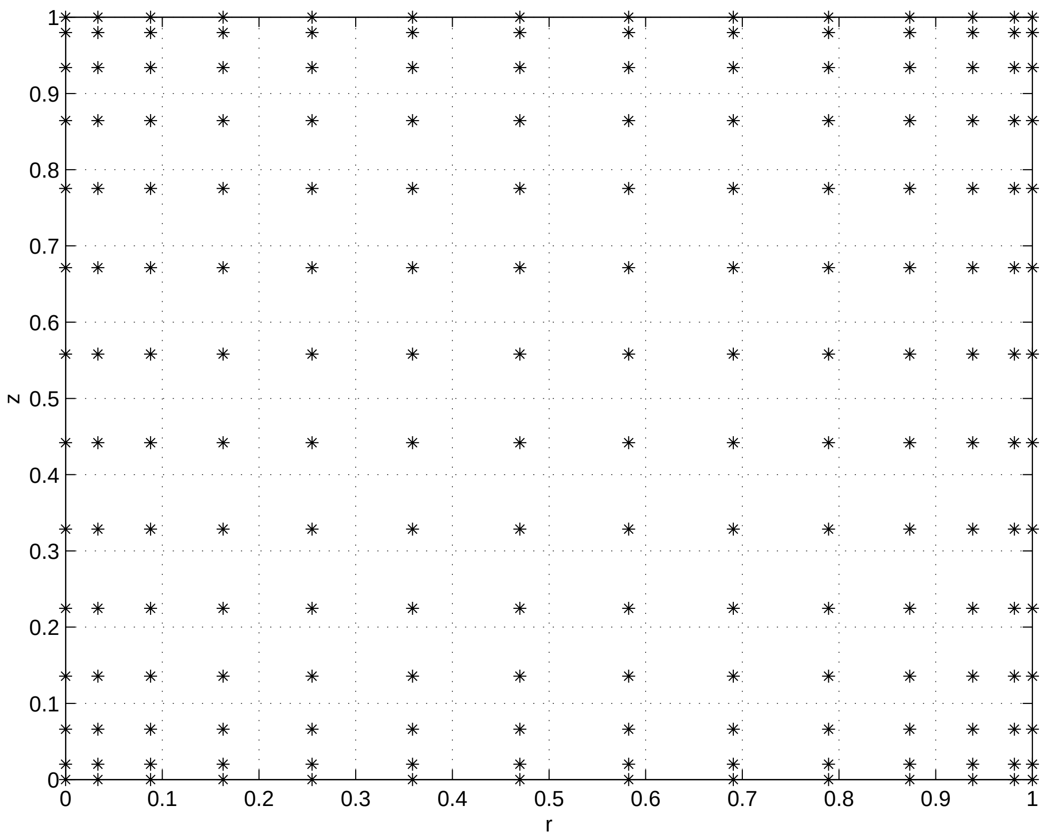

Figure 4.1: The fine-scale discretization point distribution on the physical domain.

2. We create four loper objects to store the discrete 1st-order differentiation and Laplacian operator arrays (dr, dz , ddr and ddz) :

```
DR = loper(dr, 1);
DZ = loper(dz, 2);
DDR = loper(ddr, 1);
DDZ = loper(ddz, 2);
```

4.3 Trial Functions

A time-consuming step in the implementation of Galerkin solution procedure is the actual computation of the trial functions. The trial functions are computed satisfying the following relations:

$$\lambda^r \phi = \nabla_r^2 \phi \tag{4.4}$$

$$\lambda^z \psi = \nabla_z^2 \psi \tag{4.5}$$

subject to homogeneous boundary conditions :

r=0	$\frac{d\phi}{dr} = 0$	
r=1	$\phi = 0$	
z=0		$\frac{d\psi}{dz} = 0$
z=1		$\psi = 0$

This trial function problem fits the Sturm-Liouville form [13]

$$\frac{1}{x^\alpha v(x)} \frac{d}{dx}\left(x^\alpha p(x) \frac{d\psi}{dx}\right) + q(x)\frac{d\psi}{dx} + g(x)\psi = \lambda\psi \tag{4.6}$$

for $x \in (0,1)$. The solutions $\psi(x)$ are subject to boundary conditions

$$a\frac{d\psi(0)}{dx} + b\psi(0) + a_1\frac{d\psi(1)}{dx} + b_1\psi(1) = 0 \tag{4.7}$$

$$c\frac{d\psi(1)}{dx} + d\psi(1) + c_0\frac{d\psi(0)}{dx} + d_0\psi(0) = 0 \tag{4.8}$$

Two sets of discretized, normalized trial functions are obtained after the `sl.m` function is called.

```
[lamr, phi] = sl('cyln', dr, r, 1, 0, 0, 1, wr);
[lamz, psi] = sl('slab', dz, z, 1, 0, 0, 1, wz);
```

The three trial functions along r or z direction are plotted in Figure 4.2.

A tfun object P1 is created to store the information of the two sets of trial functions ϕ and ψ.

```
P1 = tfun({phi, psi}, [1  2], {wr, wz});
```

Choosing the trial functions $\eta(r)$ and $\xi(z)$ to expand the solution of porosity ϵ is not so obvious as those chosen for $\phi(r)$ and $\psi(z)$. Because the value of ϵ at the domain boundary is not fixed, we choose $\eta(r)$ and $\xi(z)$ as polynomials that do not satisfy a pre-selected boundary condition:

$$\eta(r) = r^i \tag{4.9}$$

$$\xi(z) = z^j \tag{4.10}$$

with $i = 0, 1, \ldots, I$ and $j = 0, 1, \ldots, J$.

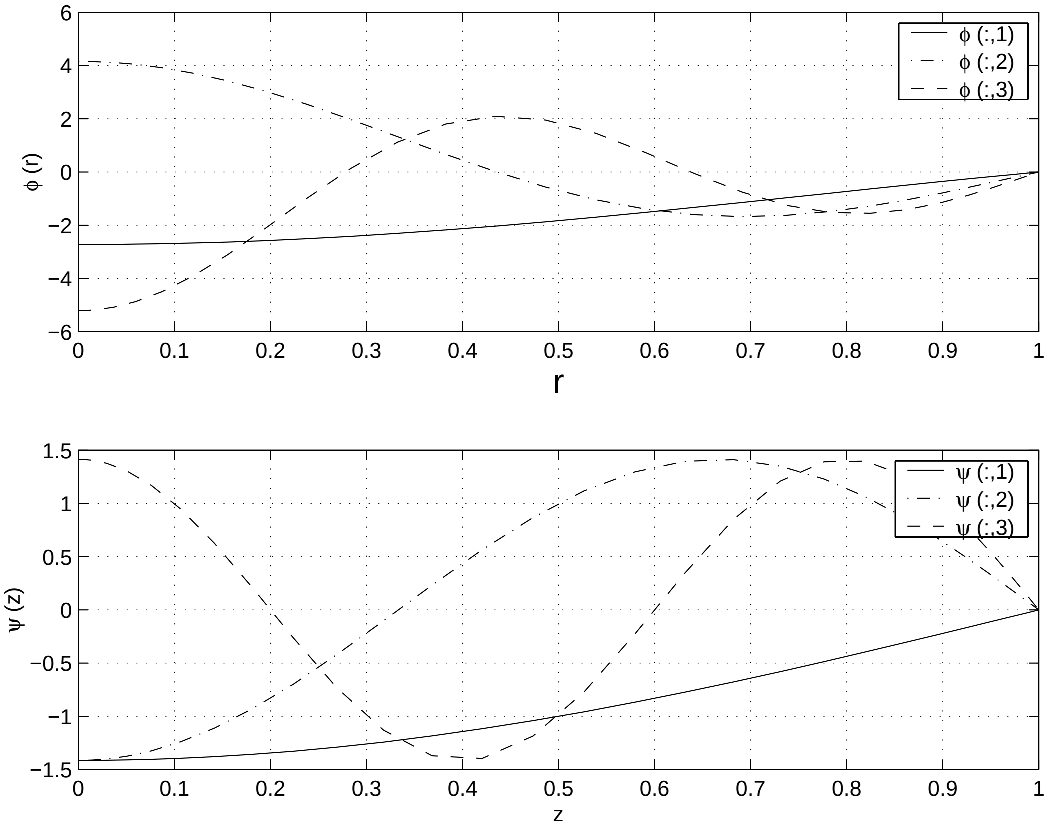

Figure 4.2: The first set of trial functions.

The MATLAB commands are

```
i = 1 : 3
    eta(:, i) = r. ∧ (i − 1);
    xi(:, i) = z. ∧ (i − 1);
end
```

Using the Gram-Schmidt orthogonalization, we make the η_i orthonormal with respect to weight wr and ξ_i orthonormal with respect to weight wz. Two functions are defined as orthonormal over a domain Ω if they satisfy

$$\langle \eta_j, \eta_k \rangle = \int_\Omega \langle \eta_j, \eta_k \rangle d\omega = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } j = k \end{cases} \tag{4.11}$$

These operations are performed with the following MWRtools functions; these trial functions are plotted in Figure 4.3 .

```
eta = gs(eta, wr);
xi = gs(xi, wz);
```

A tfun object P2 is created to store the information of the trial functions η and ξ.

```
P2 = tfun({eta, xi}, [1  2], {wr, wz});
```

4.4 Discretization of the PDE System

Solving this system using a semi-discrete projection method is not difficult. However, extending this approach to more complex systems can be much more challenging primarily due to the "book-keeping" problem. To overcome this problem, we propose to store the solution in the form of the MATLAB cell array data structures. The unknown mode amplitude coefficients **a**, **b** and **c** can be arranged in

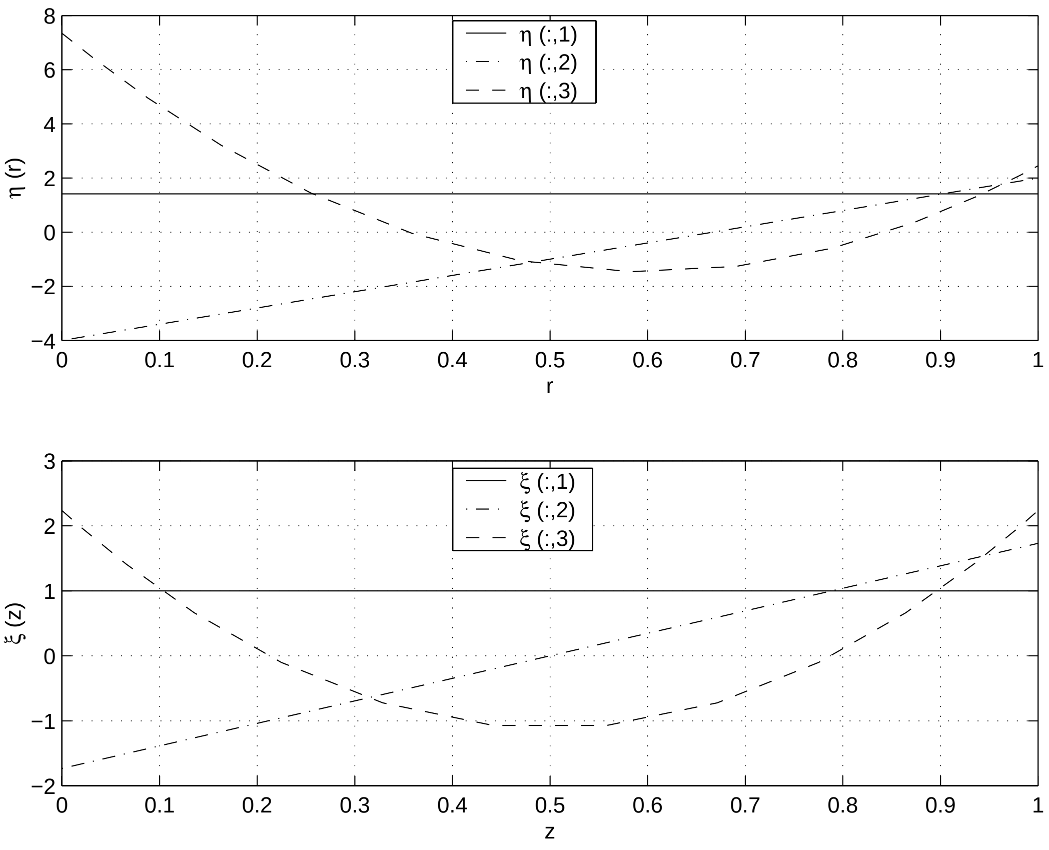

Figure 4.3: The second set of trial functions.

this way :

$$\mathbf{S}(t) = \left\{ \begin{array}{c} \begin{bmatrix} a_{1,1}(t) & a_{1,2}(t) & a_{1,3}(t) \\ a_{2,1}(t) & a_{2,2}(t) & a_{2,3}(t) \\ a_{3,1}(t) & a_{3,2}(t) & a_{3,3}(t) \end{bmatrix} \\ \\ \begin{bmatrix} b_{1,1}(t) & b_{1,2}(t) & b_{1,3}(t) \\ b_{2,1}(t) & b_{2,2}(t) & b_{2,3}(t) \\ b_{3,1}(t) & b_{3,2}(t) & b_{3,3}(t) \end{bmatrix} \\ \\ \begin{bmatrix} c_{1,1}(t) & c_{1,2}(t) & c_{1,3}(t) \\ c_{2,1}(t) & c_{2,2}(t) & c_{2,3}(t) \\ c_{3,1}(t) & c_{3,2}(t) & c_{3,3}(t) \end{bmatrix} \end{array} \right\}$$

The solution can be written in time-discretized form by extending the number of dimensions of each array by one in each array's final dimension.

To discretize the PDE system, we substitute the trial function expansion solution into the PDE equations, then project the resulting functions onto each trial function to obtain a residual array that is of the same form and size as $\mathbf{S}(t)$.

For the mass balance equation of CH_4, the semi-discrete form of Equation 3.27 becomes

$$\langle \frac{\partial C_{CH_4}}{\partial t}, \phi\psi \rangle = \langle RHS_{CH_4}, \phi\psi \rangle \tag{4.12}$$

Where RHS_{CH_4} is the right-hand-side of Equation 3.27. The left-hand-side of Equation 4.12 is

$$\langle \frac{\partial C_{CH_4}}{\partial t}, \phi\psi \rangle \quad = \quad \langle \frac{\partial}{\partial t}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z)), \phi\psi \rangle$$

$$= \quad \frac{d\mathbf{a}^{IJ}}{dt} \tag{4.13}$$

Therefore, Equation 4.12 can be rewritten as

$$\frac{d\mathbf{a}^{IJ}}{dt} = \langle RHS_{CH_4}, \phi\psi\rangle \tag{4.14}$$

For the mass balance equation of H_2, the semi-discrete form of equation (3.28) becomes

$$\langle\frac{\partial C_{H_2}}{\partial t}, \phi\psi\rangle = \langle RHS_{H_2}, \phi\psi\rangle \tag{4.15}$$

where RHS_{H_2} is the right-hand-side of equation (3.28). The left-hand-side of equation (4.15) is

$$\begin{aligned}\langle\frac{\partial C_{H_2}}{\partial t}, \phi\psi\rangle &= \langle\frac{\partial}{\partial t}(\sum_{i=1}^{I}\sum_{j=1}^{J} b_{ij}(t)\phi_i(r)\psi_j(z)), \phi\psi\rangle \\ &= \frac{d\mathbf{b}^{IJ}}{dt} \end{aligned}\tag{4.16}$$

Thus, equation (4.15) can be rewritten as

$$\frac{d\mathbf{b}^{IJ}}{dt} = \langle RHS_{H_2}, \phi\psi\rangle \tag{4.17}$$

For the governing equation of the porosity ϵ, the semi-discrete form of equation (3.26) becomes

$$\langle\frac{d\epsilon}{dt}, \eta\xi\rangle = \langle RHS_{\epsilon}, \eta\xi\rangle \tag{4.18}$$

where RHS_ϵ is the right-hand-side of equation (3.26). The left-hand-side of equation (4.18) is

$$\begin{aligned}\langle\frac{d\epsilon}{dt},\eta\xi\rangle &= \langle\frac{d}{dt}(\sum_{i=1}^{I}\sum_{j=1}^{J}c_{ij}(t)\eta_i(r)\xi_j(z)),\eta\xi\rangle \\ &= \frac{d\mathbf{c}^{IJ}}{dt} \qquad (4.19)\end{aligned}$$

Thus, equation (4.18) can be rewritten as

$$\frac{d\mathbf{c}^{IJ}}{dt} = \langle RHS_{\epsilon},\eta\xi\rangle \qquad (4.20)$$

These operations result in transferring the PDE equations to a semi-discrete ODE system which can then be written as

$$\mathbf{C}\frac{d\mathbf{S}}{dt} + \mathbf{R}(\mathbf{S},t) = 0 \qquad (4.21)$$

where $\mathbf{C}$ is the capacitance cell array created by the MATLAB command:

`C = mdiag(ones(3,3); ones(3,3); ones(3,3));` It has the form

$$\mathbf{C} = \left\{\begin{array}{c} \left\{\begin{array}{ccc} \{C^a_{1,1}\} & \{C^a_{1,2}\} & \{C^a_{1,3}\} \\ \{C^a_{2,1}\} & \{C^a_{2,2}\} & \{C^a_{2,3}\} \\ \{C^a_{3,1}\} & \{C^a_{3,2}\} & \{C^a_{3,3}\} \end{array}\right\} \\ \\ \left\{\begin{array}{ccc} \{C^b_{1,1}\} & \{C^b_{1,2}\} & \{C^b_{1,3}\} \\ \{C^b_{2,1}\} & \{C^b_{2,2}\} & \{C^b_{2,3}\} \\ \{C^b_{3,1}\} & \{C^b_{3,2}\} & \{C^b_{3,3}\} \end{array}\right\} \\ \\ \left\{\begin{array}{ccc} \{C^c_{1,1}\} & \{C^c_{1,2}\} & \{C^c_{1,3}\} \\ \{C^c_{2,1}\} & \{C^c_{2,2}\} & \{C^c_{2,3}\} \\ \{C^c_{3,1}\} & \{C^c_{3,2}\} & \{C^c_{3,3}\} \end{array}\right\} \end{array}\right\}$$

where, for example

$$\left\{C^a_{1,1}\right\} = \left\{\begin{array}{c} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}\right\}, \quad \left\{C^a_{1,2}\right\} = \left\{\begin{array}{c} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}\right\}, \quad \text{etc.}$$

$$\mathbf{R}(t) = \left\{\begin{array}{c} \begin{bmatrix} R^a_{1,1}(t) & R^a_{1,2}(t) & R^a_{1,3}(t) \\ R^a_{2,1}(t) & R^a_{2,2}(t) & R^a_{2,3}(t) \\ R^a_{3,1}(t) & R^a_{3,2}(t) & R^a_{3,3}(t) \end{bmatrix} \\ \\ \begin{bmatrix} R^b_{1,1}(t) & R^b_{1,2}(t) & R^b_{1,3}(t) \\ R^b_{2,1}(t) & R^b_{2,2}(t) & R^b_{2,3}(t) \\ R^b_{3,1}(t) & R^b_{3,2}(t) & R^b_{3,3}(t) \end{bmatrix} \\ \\ \begin{bmatrix} R^c_{1,1}(t) & R^c_{1,2}(t) & R^c_{1,3}(t) \\ R^c_{2,1}(t) & R^c_{2,2}(t) & R^c_{2,3}(t) \\ R^c_{3,1}(t) & R^c_{3,2}(t) & R^c_{3,3}(t) \end{bmatrix} \end{array}\right\}.$$

Elements of this cell array include the projection of the modeling equations

$$R^a_{p,q}(t) = -\langle RHS_{CH_4}, \phi_p \psi_q \rangle \tag{4.22}$$

$$R^b_{p,q}(t) = -\langle RHS_{H_2}, \phi_p \psi_q \rangle \tag{4.23}$$

$$R^c_{p,q}(t) = -\langle RHS_{\epsilon}, \eta_p \xi_q \rangle \tag{4.24}$$

To compute the values of $R^a_{p,q}(t)$, $R^b_{p,q}(t)$ and $R^c_{p,q}(t)$, we must calculate RHS_{CH_4}, RHS_{H_2} and RHS_{ϵ}. The programming details for implementation of this procedure in the MATLAB environment are given in Figure 4.4, 4.5 and 4.6. `c1`, `c2` and `e` are used to represent C_{CH_4}, C_{H_2} and ϵ. We project RHS_{CH_4}, RHS_{H_2} and RHS_{ϵ} onto a tfun object storing the information of the trial functions by calling the `wip.m` method of the tfun class, respectively.

$\mathtt{R^a = -wip(RHS_{CH_4}, P1);}$

$\mathtt{R^b = -wip(RHS_{H_2}, P1);}$

$\mathtt{R^c = -wip(RHS_{\epsilon}, P2);}$

4.5 Initial Conditions

At the initial state ($t = 0$), the concentration of CH_4 is set to zero. Therefore

$$C_{CH_4}(r, z, t=0) = C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t=0)\phi_i(r)\psi_j(z) = 0$$

$$\sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t=0)\phi_i(r)\psi_j(z) = -C^R_{CH_4}$$

We project both sides of the above equation onto the trial functions ϕ and ψ

RHS terms of Eq. 3.27	Equivalent MATLAB code
$\frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial C_{CH_4}}{\partial r})$	`alphar * (DDR * c1)`
$\frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial C_{CH_4}}{\partial r}$	`(alphar * (DR * e). * (DR * c1))./e`
$\alpha_z\frac{\partial^2 C_{CH_4}}{\partial z^2}$	`alphaz * (DDZ * c1)`
$\frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial C_{CH_4}}{\partial z}$	`(alphaz * (DZ * e). * (DZ * c1))./e`
ζC_{CH_4}	`zeta * c1`
$\gamma\zeta C_{CH_4}^2$	`gamma * zeta * c1.^2`

Figure 4.4: Comparison of the CH_4 residual function terms to the corresponding computational steps of Object-Oriented projection methods.

$$\langle\sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t=0)\phi_i(r)\psi_j(z), \phi\psi\rangle = \langle -C_{CH_4}^R, \phi\psi\rangle$$

Because both ϕ and ψ are orthonormal functions, we have

$$\langle\phi_p\psi_q, \phi_i\psi_j\rangle = 0 \quad (p \neq i, q \neq j) \tag{4.25}$$

Thus

$$\mathbf{a}^{IJ}(t=0) = \langle -C_{CH_4}^R, \phi\psi\rangle \tag{4.26}$$

The concentration of H_2 at the initial state is also set to zero, and so from Equation 4.2, we have

$$C_{H_2}(r,z,t=0) = \sum_{i=1}^{I}\sum_{j=1}^{J} b_{ij}(t=0)\phi_i(r)\psi_j(z) = 0$$

RHS terms of Eq. 3.28	Equivalent MATLAB code
$\frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial C_{H_2}}{\partial r})$	`alphar * (DDR * c2)`
$\frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial C_{H_2}}{\partial r}$	`(alphar * (DR * e). * (DR * c2))./e`
$\alpha_z\frac{\partial^2 C_{H_2}}{\partial z^2}$	`alphaz * (DDZ * c2)`
$\frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial C_{H_2}}{\partial z}$	`(alphaz * (DZ * e). * (DZ * c2))./e`
$2\zeta C_{CH_4}$	`2 * zeta * c1`
$\gamma\zeta C_{CH_4}C_{H_2}$	`gamma * zeta * c1. * c2`

Figure 4.5: Comparison of the H_2 residual function terms to the corresponding computational steps of Object-Oriented projection methods.

RHS terms of Equation 3.26	Equivalent MATLAB code
$\gamma\zeta C_{CH_4}\epsilon$	`gamma * zeta * c1. * e`

Figure 4.6: Comparison of the ϵ residual function term to the corresponding computational steps of Object-Oriented projection methods.

Both sides of the above equation are projected onto the orthonormal trial functions ϕ and ψ

$$\langle \sum_{i=1}^{I}\sum_{j=1}^{J} b_{ij}(t=0)\phi_i(r)\psi_j(z), \phi\psi\rangle = \langle 0, \phi\psi\rangle$$

$$\mathbf{b}^{IJ}(t=0) = 0 \tag{4.27}$$

The initial porosity ϵ is set to a constant value ϵ_0 ($\epsilon_0 = 0.7$) as same as in [20], from trial function expansion equation 4.3,

$$\epsilon(r,z,t=0) = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}(t=0)\eta_i(r)\xi_j(z) = \epsilon_0$$

The above equation is projected onto orthonormal trial functions η and ξ.

$$\langle \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}(t=0)\eta_i(r)\xi_j(z), \eta\xi\rangle = \langle \epsilon_0, \eta\xi\rangle$$

Hence

$$\mathbf{c}^{IJ}(t=0) = \langle \epsilon_0, \eta\xi\rangle \tag{4.28}$$

The MWRtools function `wip.m` is called to implement these calculations

$\mathtt{a}_0 = \mathtt{wip}(\mathtt{repmat}(-\mathtt{C}^{\mathtt{R}}_{\mathtt{CH_4}}, 14, 14), \mathtt{P1});$

$\mathtt{b}_0 = \mathtt{zeros}(3, 3);$

$\mathtt{c}_0 = \mathtt{wip}(\mathtt{repmat}(\epsilon_0, 14, 14), \mathtt{P2});$

4.6 Integration of the ODE Set

The solution S can be rewritten in time-discretized form by extending the number of dimensions of each array by one. For example, the two dimensional coefficient array $\mathbf{a}^{IJ}$ will be a three dimensional array $\mathbf{a}^{IJK}$ now. The last dimension denotes the discrete points in time t_k, $k = 1, \ldots, K$.

A ordinary differential/algebraic equations solver `odaepc.m` is developed based on the method of orthogonal-collocation described by Villadsen and Stewart [29]. The derivative of a function y is given by

$$\frac{dy}{dx} \mid_{(x_i)} = \sum_{j=1}^{N} \mathbf{A}_{\mathbf{x}}(i,j) y(x_j) \tag{4.29}$$

If $\mathbf{A}$ is the $(K \times K)$ discrete time-differentiation array, the problem to be solved becomes

$$\mathcal{C}[\mathbf{A}\mathcal{S}]_k + \mathbf{R}(\mathcal{S}) = 0 \tag{4.30}$$

where $[\mathbf{A}\mathcal{S}]_k$ denotes discrete time-differentiation operation, operating on the last non-singleton dimension of each matrix making up the cell array $\mathcal{S}$. In the discretized-time case, the capacitance cell array structure is defined so that the product $\mathcal{C}\mathcal{S}$ produces the result equivalent to $\mathbf{C}\mathbf{S}(t_k)$. Therefore, we can define the corresponding residual cell array and Jacobian cell and solve this system using the Newton-Raphson iterations, taking into account the initial conditions.

4.6.1 Implementation

The numerical procedures are implemented with the following steps:

1. Constant model parameters and the basis functions for the spatial discretizations are set up and stored in a cell array.

2. Parameters and forcing functions that are explicit functions of time are defined at the collocation points in time and are stored in the cell array

`ffun`. When `odaepc` is called, the forcing functions in time are spectrally filtered, if required.

3. The "right-hand-sides" of the ordinary differential/algebraic modeling equations are defined in a MATLAB function similar in structure to what is required for a standard ODE solver. The format is slightly different to account for the cell array format. For example, in a problem consisting of three different modeling equations, after discretization, the residual functions ($\mathbf{R^a}$, $\mathbf{R^b}$, and $\mathbf{R^c}$) in time resulting from the projection operations would be arranged as

$$\mathbf{R}(t) = \left\{ \begin{array}{c} -[\mathbf{R^a}] \\ \\ -[\mathbf{R^b}] \\ \\ -[\mathbf{R^c}] \end{array} \right\}.$$

4. The capacitance cell array $\mathbf{C}$ or the template cell $\mathbf{T}$ must be specified; if $\mathbf{T}$ is specified, $\mathbf{C}$ is generated as the diagonal equivalent to $\mathbf{T}$ using `mdiag.m`.

5. After the transient solution is computed, slices in time can be extracted using `extract.m`. Steady state solutions can also be computed using `msolve.m`.

4.6.2 Jacobian Array

If the problem has three state variables in discretized form ($\mathbf{u^a}$, $\mathbf{u^b}$,$\mathbf{u^c}$), the Jacobian array elements must be supplied in the form

$$\mathbf{J_{temp}} = \left\{ \begin{array}{c} \left\{ \begin{array}{c} \left\{\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^a}}\right\} \\ \left\{\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^b}}\right\} \\ \left\{\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^c}}\right\} \end{array} \right\} \\ \\ \left\{ \begin{array}{c} \left\{\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^a}}\right\} \\ \left\{\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^b}}\right\} \\ \left\{\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^c}}\right\} \end{array} \right\} \\ \\ \left\{ \begin{array}{c} \left\{\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^a}}\right\} \\ \left\{\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^b}}\right\} \\ \left\{\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^c}}\right\} \end{array} \right\} \end{array} \right\}$$

In this work, the cell elements in the above equation is calculated in this way:

1. Substitute the trial function expansion into the modeling equations 3.27, 3.28 and 3.17.
2. Find the derivatives of the right-hand-side of the modeling equations with respect to the coefficients.
3. Project the derivatives onto the trial functions.

The calculation of $\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^a}}$ is shown below

$$\frac{d}{da}(RHS_{CH_4}) \quad = \quad \frac{\partial}{\partial a}(\frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z))))$$

$$
\begin{aligned}
&+ \quad \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial}{\partial a}\frac{\partial}{\partial r}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z)) \\
&+ \quad \alpha_z \frac{\partial}{\partial a}(\frac{\partial^2}{\partial z^2}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z))) \\
&+ \quad \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial}{\partial a}\frac{\partial}{\partial z}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z)) \\
&- \quad \zeta \frac{\partial}{\partial a}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z)) \\
&+ \quad \gamma\zeta \frac{\partial}{\partial a}(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z))^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{d}{da}(RHS_{CH_4}) \quad &= \quad \frac{\alpha_r}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)) + \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)) \\
&+ \quad \alpha_z \frac{\partial^2}{\partial z^2}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)) + \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)) \\
&- \quad \zeta \sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z) \\
&+ \quad 2\gamma\zeta(C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}(t)\phi_i(r)\psi_j(z))\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)
\end{aligned}
$$

Finally, we project the above equation onto the trial functions $\phi\psi$:

$$
\langle\frac{d}{da}(RHS_{CH_4}), \phi\psi\rangle \quad = \quad \alpha_r\langle\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle
$$

$$
\begin{aligned}
&+ \quad \langle \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle \\
&+ \quad \alpha_z\langle \frac{\partial^2}{\partial z^2}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle \\
&+ \quad \langle \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle \\
&- \quad \zeta\langle \sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle \\
&+ \quad 2\gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle \qquad (4.31)
\end{aligned}
$$

The corresponding programming implementation in MATLAB is shown in Figure 4.7.

Follow the similar computing procedure, we can obtain the other cell elements in the Jacobian array.

$$
\begin{aligned}
\langle \frac{d}{dc}(RHS_{CH_4}), \phi\psi\rangle \quad &= \quad \langle \frac{\alpha_r}{\epsilon}\frac{\partial C_{CH_4}}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle \\
&+ \quad \langle \frac{\alpha_r}{\epsilon^2}\frac{\partial C_{CH_4}}{\partial r}\frac{\partial \epsilon}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle \\
&+ \quad \langle \frac{\alpha_z}{\epsilon}\frac{\partial C_{CH_4}}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle \\
&+ \quad \langle \frac{\alpha_z}{\epsilon^2}\frac{\partial C_{CH_4}}{\partial z}\frac{\partial \epsilon}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle \qquad (4.32)
\end{aligned}
$$

Terms in $\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^a}}$	Equivalent MATLAB code
$\alpha_r\langle\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphar * wip(DDR * P1, P1)`
$\langle\frac{\alpha_r}{\epsilon}\frac{\partial\epsilon}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphar * wip(((DR * e)./e). * (DR * P1), P1)`
$\alpha_z\langle\frac{\partial^2}{\partial z^2}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphaz * wip(DDZ * P1, P1)`
$\langle\frac{\alpha_z}{\epsilon}\frac{\partial\epsilon}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphaz * wip(((DZ * e)./e). * (DZ * P1), P1)`
$\zeta\langle\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle$	`zeta * wip(P1, P1)`
$2\gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle$	`2 * gamma * zeta * wip(c1. * P1, P1)`

Figure 4.7: Comparison of the cell element $\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^a}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

Terms in $\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^c}}$	Equivalent MATLAB code
$\langle \frac{\alpha_r}{\epsilon}\frac{\partial C_{CH_4}}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle$	`alphar*wip(((DR*c1)./e).*(DR*P2),P1)`
$\langle \frac{\alpha_r}{\epsilon^2}\frac{\partial C_{CH_4}}{\partial r}\frac{\partial \epsilon}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle$	`alphar*wip(((DR*e).*(DR*c1)./(e.^2)).*P2,P1)`
$\langle \frac{\alpha_z}{\epsilon}\frac{\partial C_{CH_4}}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle$	`alphaz*wip(((DZ*c1)./e).*(DZ*P2),P1)`
$\langle \frac{\alpha_z}{\epsilon^2}\frac{\partial C_{CH_4}}{\partial z}\frac{\partial \epsilon}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle$	`alphaz*wip(((DZ*e).*(DZ*c1)./(e.^2)).*P2,P1)`

Figure 4.8: Comparison of the cell element $\frac{\partial \mathbf{R^a}}{\partial \mathbf{u^c}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

$$\begin{aligned}\langle \frac{d}{da}(RHS_{H_2}),\phi\psi\rangle &= \gamma\zeta\langle C_{H_2}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle \\ &+ 2\zeta\langle\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle \qquad (4.33)\end{aligned}$$

$$\begin{aligned}\langle \frac{d}{db}(RHS_{H_2}),\phi\psi\rangle &= \alpha_r\langle\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)),\phi\psi\rangle \\ &+ \langle\frac{\alpha_r}{\epsilon}\frac{\partial\epsilon}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)),\phi\psi\rangle \\ &+ \alpha_z\langle\frac{\partial^2}{\partial z^2}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)),\phi\psi\rangle\end{aligned}$$

Terms in $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^a}}$	Equivalent MATLAB code
$\gamma\zeta\langle C_{H_2}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle$	`gamma * zeta * wip(c2. * P1, P1)`
$2\zeta\langle\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle$	`2 * zeta * wip(P1, P1)`

Figure 4.9: Comparison of the cell element $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^a}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

$$\begin{aligned}
& + \quad \langle\frac{\alpha_z}{\epsilon}\frac{\partial\epsilon}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)),\phi\psi\rangle \\
& + \quad 2\zeta\langle\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle \\
& + \quad \gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\phi\psi\rangle \qquad (4.34)
\end{aligned}$$

$$\begin{aligned}
\langle\frac{d}{dc}(RHS_{H_2}),\phi\psi\rangle &= \langle\frac{\alpha_r}{\epsilon}\frac{\partial C_{H_2}}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle \\
&+ \langle\frac{\alpha_r}{\epsilon^2}\frac{\partial C_{H_2}}{\partial r}\frac{\partial\epsilon}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle \\
&+ \langle\frac{\alpha_z}{\epsilon}\frac{\partial C_{H_2}}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)),\phi\psi\rangle
\end{aligned}$$

Terms in $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^b}}$	Equivalent MATLAB code
$\alpha_r \langle \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphar * wip(DDR * P1, P1)`
$\langle \frac{\alpha_r}{\epsilon}\frac{\partial \epsilon}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphar * wip(((DR * e)./e). * (DR * P1), P1)`
$\alpha_z \langle \frac{\partial^2}{\partial z^2}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphaz * wip(DDZ * P1, P1)`
$\langle \frac{\alpha_z}{\epsilon}\frac{\partial \epsilon}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z)), \phi\psi\rangle$	`alphaz * wip(((DZ * e)./e). * (DZ * P1), P1)`
$2\zeta\langle\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle$	`2zeta * wip(P1, P1)`
$\gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \phi\psi\rangle$	`gamma * zeta * wip(c1. * P1, P1)`

Figure 4.10: Comparison of the cell element $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^b}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

$$+ \quad \langle \frac{\alpha_z}{\epsilon^2}\frac{\partial C_{H_2}}{\partial z}\frac{\partial \epsilon}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle \tag{4.35}$$

Terms in $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^c}}$	Equivalent MATLAB code
$\langle \frac{\alpha_r}{\epsilon}\frac{\partial C_{H_2}}{\partial r}\frac{\partial}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle$	`alphar * wip(((DR * c2)./e). * (DR * P2), P1)`
$\langle \frac{\alpha_r}{\epsilon^2}\frac{\partial C_{H_2}}{\partial r}\frac{\partial \epsilon}{\partial r}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle$	`alphar * wip(((DR * e). * (DR * c2)./(e.^2)). * P2, P1)`
$\langle \frac{\alpha_z}{\epsilon}\frac{\partial C_{H_2}}{\partial z}\frac{\partial}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle$	`alphaz * wip(((DZ * c2)./e). * (DZ * P2), P1)`
$\langle \frac{\alpha_z}{\epsilon^2}\frac{\partial C_{H_2}}{\partial z}\frac{\partial \epsilon}{\partial z}(\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z)), \phi\psi\rangle$	`alphaz * wip(((DZ * e). * (DZ * c2)./(e.^2)). * P2, P1)`

Figure 4.11: Comparison of the cell element $\frac{\partial \mathbf{R^b}}{\partial \mathbf{u^c}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

$$\langle \frac{d}{da}(RHS_\epsilon), \eta\xi\rangle = \gamma\zeta\langle \epsilon\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z), \eta\xi\rangle \tag{4.36}$$

$$\langle \frac{d}{dc}(RHS_\epsilon), \eta\xi\rangle = \gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z), \eta\xi\rangle \tag{4.37}$$

The cell array structure $\mathbf{J_{temp}}$ is passed to the function `makejacobian.m`, to create $\mathbf{J}$ in the proper format: a long column of column cell arrays that has the same structure as $\mathbf{C}$.

Terms in $\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^a}}$	Equivalent MATLAB code
$\gamma\zeta\langle\epsilon\sum_{i=1}^{I}\sum_{j=1}^{J}\phi_i(r)\psi_j(z),\eta\xi\rangle$	`gamma * zeta * wip(e. * P1, P2)`

Figure 4.12: Comparison of the cell element $\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^a}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

Terms in $\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^c}}$	Equivalent MATLAB code
$\gamma\zeta\langle C_{CH_4}\sum_{i=1}^{I}\sum_{j=1}^{J}\eta_i(r)\xi_j(z),\eta\xi\rangle$	`gamma * zeta * wip(c1. * P2, P2)`

Figure 4.13: Comparison of the cell element $\frac{\partial \mathbf{R^c}}{\partial \mathbf{u^c}}$ in the Jacobian array to the corresponding computational steps used in Object-Oriented numerical approach.

$$\mathbf{J} = \left\{ \begin{array}{c} \left\{ \begin{array}{ccc} \left\{J^a_{1,1}\right\} & \left\{J^a_{1,2}\right\} & \left\{J^a_{1,3}\right\} \\ \left\{J^a_{2,1}\right\} & \left\{J^a_{2,2}\right\} & \left\{J^a_{2,3}\right\} \\ \left\{J^a_{3,1}\right\} & \left\{J^a_{3,2}\right\} & \left\{J^a_{3,3}\right\} \end{array} \right\} \\ \\ \left\{ \begin{array}{ccc} \left\{J^b_{1,1}\right\} & \left\{J^b_{1,2}\right\} & \left\{J^b_{1,3}\right\} \\ \left\{J^b_{2,1}\right\} & \left\{J^b_{2,2}\right\} & \left\{J^b_{2,3}\right\} \\ \left\{J^b_{3,1}\right\} & \left\{J^b_{3,2}\right\} & \left\{J^b_{3,3}\right\} \end{array} \right\} \\ \\ \left\{ \begin{array}{ccc} \left\{J^c_{1,1}\right\} & \left\{J^c_{1,2}\right\} & \left\{J^c_{1,3}\right\} \\ \left\{J^c_{2,1}\right\} & \left\{J^c_{2,2}\right\} & \left\{J^c_{2,3}\right\} \\ \left\{J^c_{3,1}\right\} & \left\{J^c_{3,2}\right\} & \left\{J^c_{3,3}\right\} \end{array} \right\} \end{array} \right\}$$

and where, for example

$$\left\{J^a_{m,n}\right\} = \left\{ \begin{array}{c} \left[\frac{\partial R^a_{i,j}}{\partial a_{m,n}}\right] \\ \\ \left[\frac{\partial R^b_{i,j}}{\partial a_{m,n}}\right] \\ \\ \left[\frac{\partial R^c_{i,j}}{\partial a_{m,n}}\right] \end{array} \right\}$$

Chapter 5

Results and Discussion

5.1 Solution Construction

After convergence of the collocation-based time integration, the solution cell array **S** will contain three matrices defining the mode amplitude coefficients **a**, **b** and **c**. Each is a three dimensional matrix with the last dimension defining the discrete points in time. To extract the solution at a specific time t_k, we call the MATLAB function `extract.m`

```
solution = extract(S,k);
```

Having introduced the concept of Object-Oriented programming, the reconstruction of spectrally discretized solutions becomes much more straightforward. Figure 5.1 shows the commands to compute the value of C_{CH_4}, C_{H_2} and ϵ.

5.2 Discussion of Results

Figures 5.2, 5.3, 5.4 show the evolution of CH_4, H_2 concentration and preform porosity during isothermal CVI at a temperature of 1400°K and a pressure of 100 Torr. Densification proceeds in an "outside-in" pattern with a faster rate

Solution in function expansion form	Equivalent MATLAB code
$C_{CH_4} = C^R_{CH_4} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}\phi_i\psi_j$	$\mathtt{C_{CH_4} = C^R_{CH_4} + a * P3}$
$C_{H_2} = \sum_{i=1}^{I}\sum_{j=1}^{J} b_{ij}\phi_i\psi_j$	$\mathtt{C_{H_2} = b * P3}$
$\epsilon = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}\eta_i\xi_j$	$\epsilon = \mathtt{c * P4}$

Figure 5.1: Comparison of the trial function expansion form of the value of C_{CH_4}, C_{H_2} and ϵ to the corresponding computations using of Object-Oriented methods.

of densification at the edges than at the center of the preform. We can see the maximum value of porosity is at the center of the preform. This densification pattern is due to mass transfer limitations of the gas within the preform. As the exterior of the preform densifies, diffusional resistance for the precursor gas increases, leading to the development of a relatively porous region in the center of the preform. The initial porosity is set to be 0.7 and it drops to approximately 0.12 after 27 hours. The concentration of CH_4 has its maximum value at the surface of the preform. After it diffuses into the preform, chemical reaction takes place and H_2 is produced. Thus, H_2 has a negative concentration gradient from the center of the preform toward the surface.

The choice of truncation number is a very important issue in the application of spectral methods. In general, accuracy should increase with higher truncation numbers. However, computing time will increase dramatically as the number of trial functions grows. Figures 5.5, 5.6, 5.7 show the evolution of the concentra-

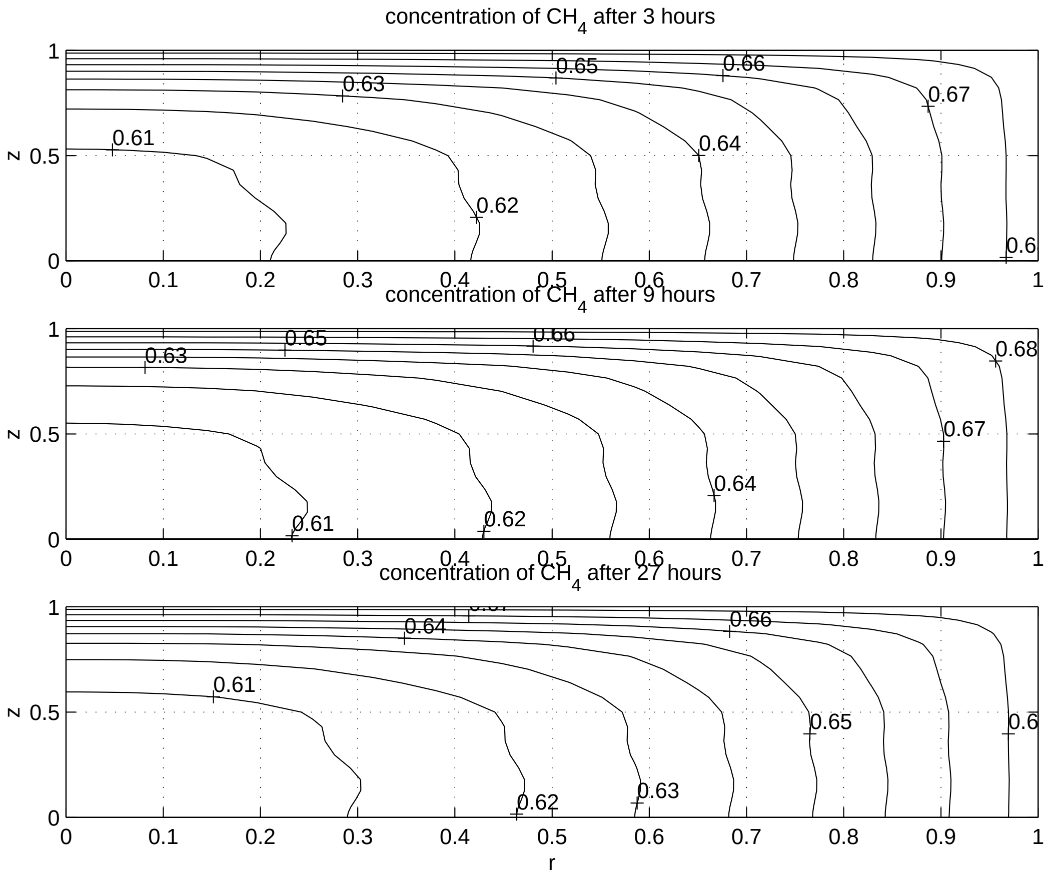

Figure 5.2: Evolution of CH_4 concentration at 3, 9, 27 hr. Temperature is 1400°K and operating pressure is 100 Torr.

tion of CH_4, H_2 and the porosity with different truncation numbers. Generally, when the number of trial functions increases, the number of fine-grid discretized points also must increase. In this work, 14 discretized points are used for 3 trial functions, 23 discretized points are used for 6 trial functions, and 30 discretized points are used for 10 trial functions. From the figures, we can see that the shape of the solution curves changes significantly as the number of trial functions increases, especially when it increases from 3 to 6. When only 3 trial functions are used, acceptable convergence of the porosity at the domain boundary $z = 0$ can

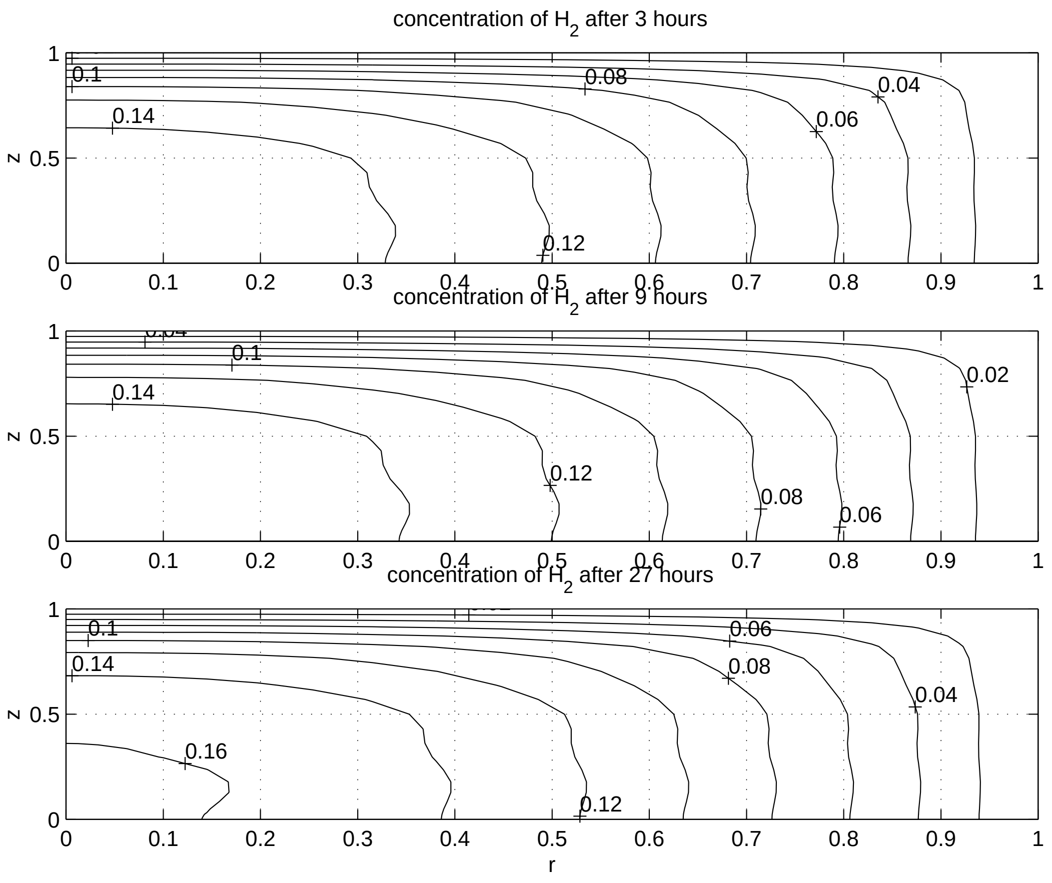

Figure 5.3: Evolution of H_2 concentration at 3, 9, 27 hr. Temperature is 1400°K and operating pressure is 100 Torr.

not be reached. Because the polynomials z^i were chosen as the trial functions, when $i = 1$, the derivative of the trial function with respect to z will not vanish at $z = 0$. However, when the truncation number of trial functions is increased, the spectral method will reduce the influence of the odd trial functions on the converged solution. The resulting curves will become smoother and more regular.

Time integrator resolution is another important issue that will affect the accuracy of the numerical techniques. When the integrator `odaepc.m` is called, the orthogonal collocation method is used. Figures 5.8, 5.9 and 5.10 show the

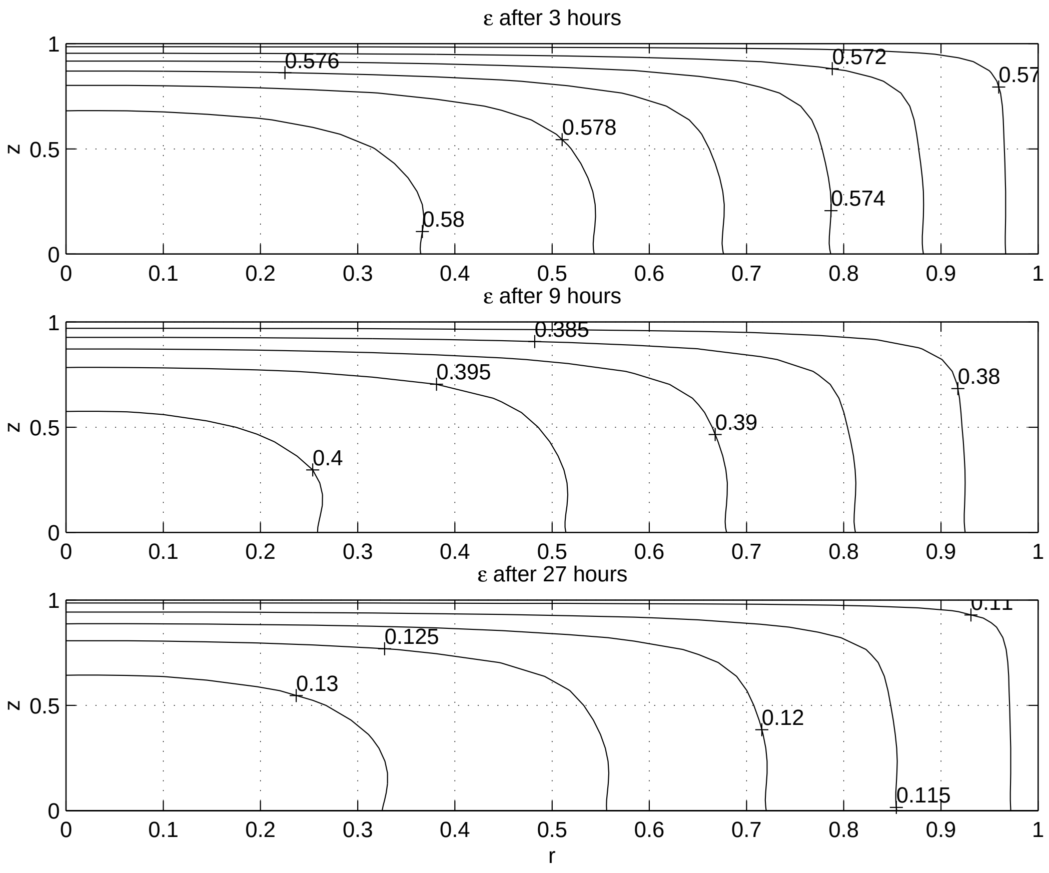

Figure 5.4: Evolution of porosity at 3, 9, 27 hr. Temperature is 1400°K and operating pressure is 100 Torr.

evolution of CH_4, H_2 concentration and porosity at $t = 3$ hours with different collocation points in time. The temperature is 1400°K and the pressure is 100 Torr. Little difference can be observed from the figures when the number of collocation points in time is changed from 2 to 6.

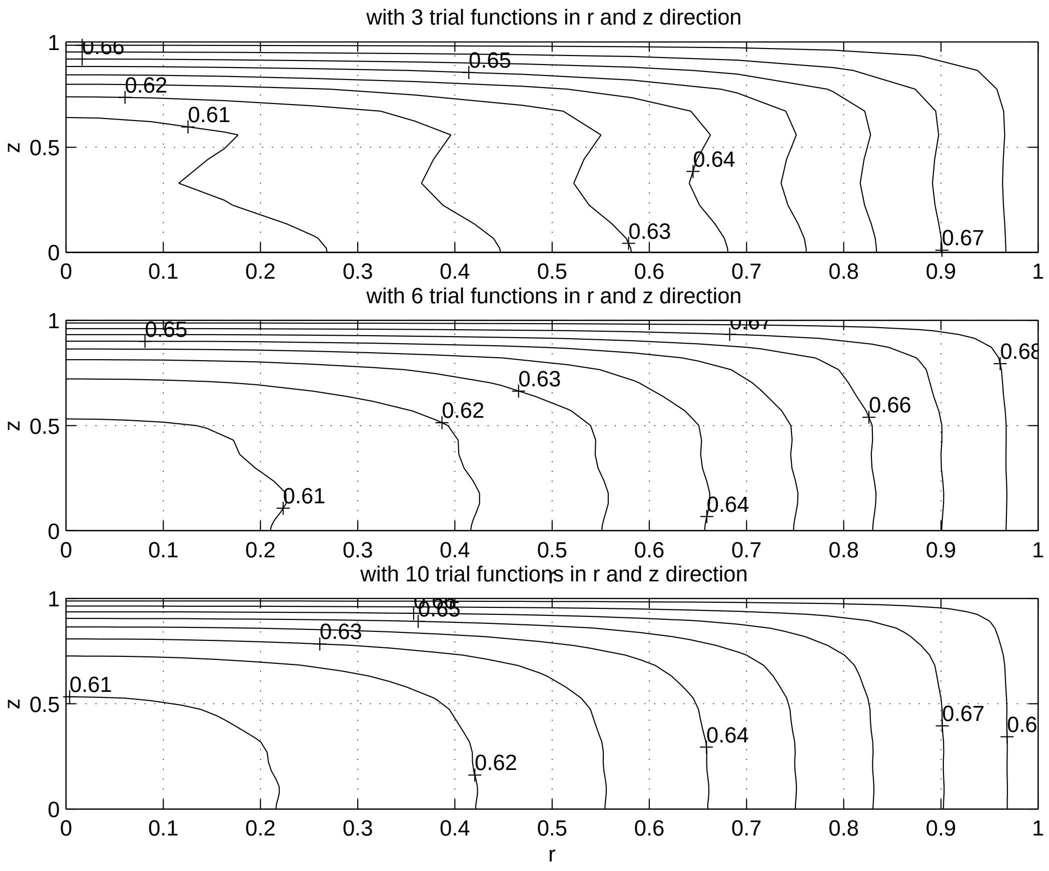

Figure 5.5: Evolution of CH_4 concentration with 3, 6, 10 trial functions along each direction (r and z). Temperature is 1400°K and operating pressure is 100 Torr. $t = 3$hr. The number of collocation point in time is set at 6.

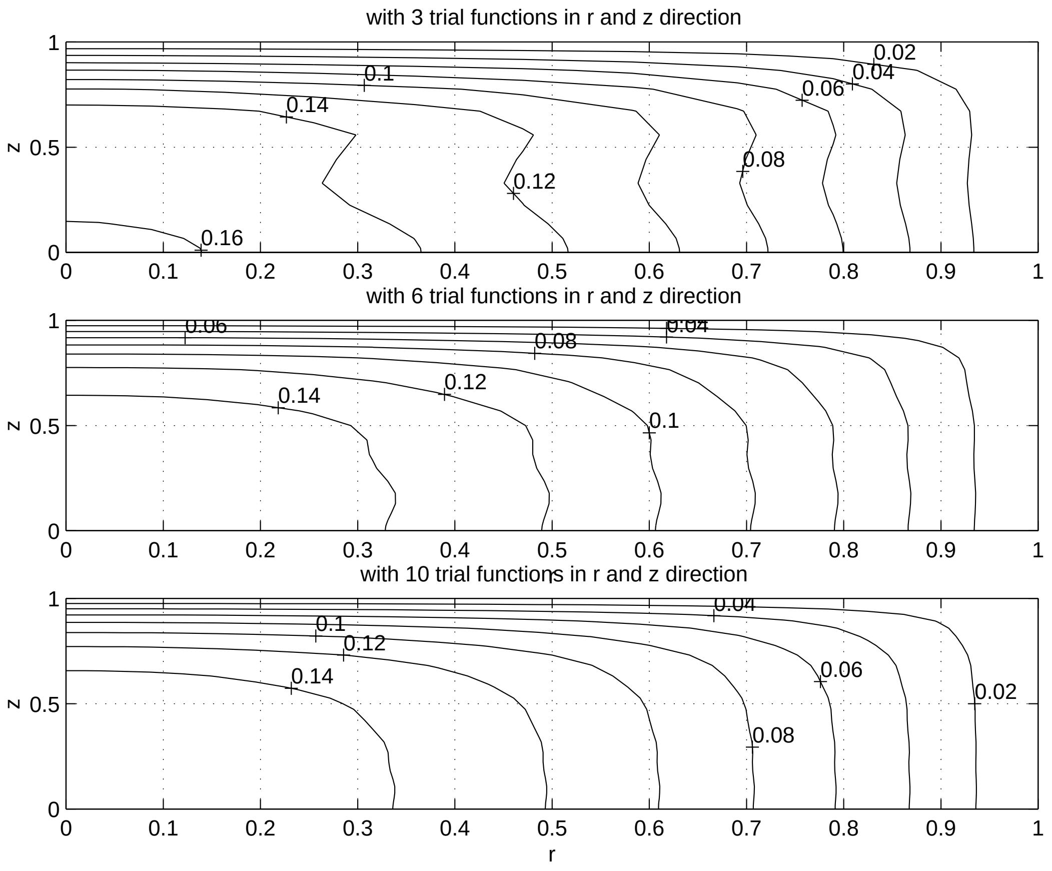

Figure 5.6: Evolution of H_2 concentration with 3, 6, 10 trial functions along each direction (r and z). Temperature is 1400°K and operating pressure is 100 Torr. $t = 3$hr. The number of collocation point in time is set at 6.

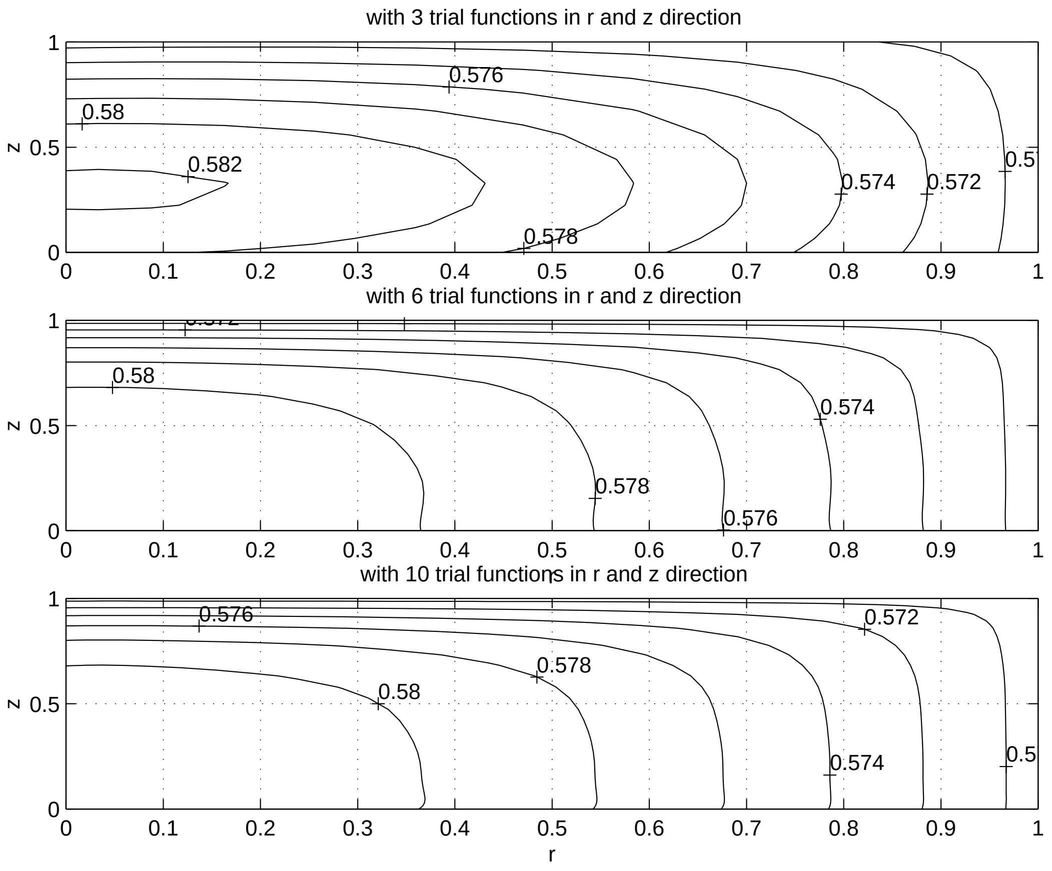

Figure 5.7: Evolution of porosity with 3, 6, 10 trial functions along each direction (r and z). Temperature is 1400°K and operating pressure is 100 Torr. $t = 3$hr. The number of collocation point in time is set at 6.

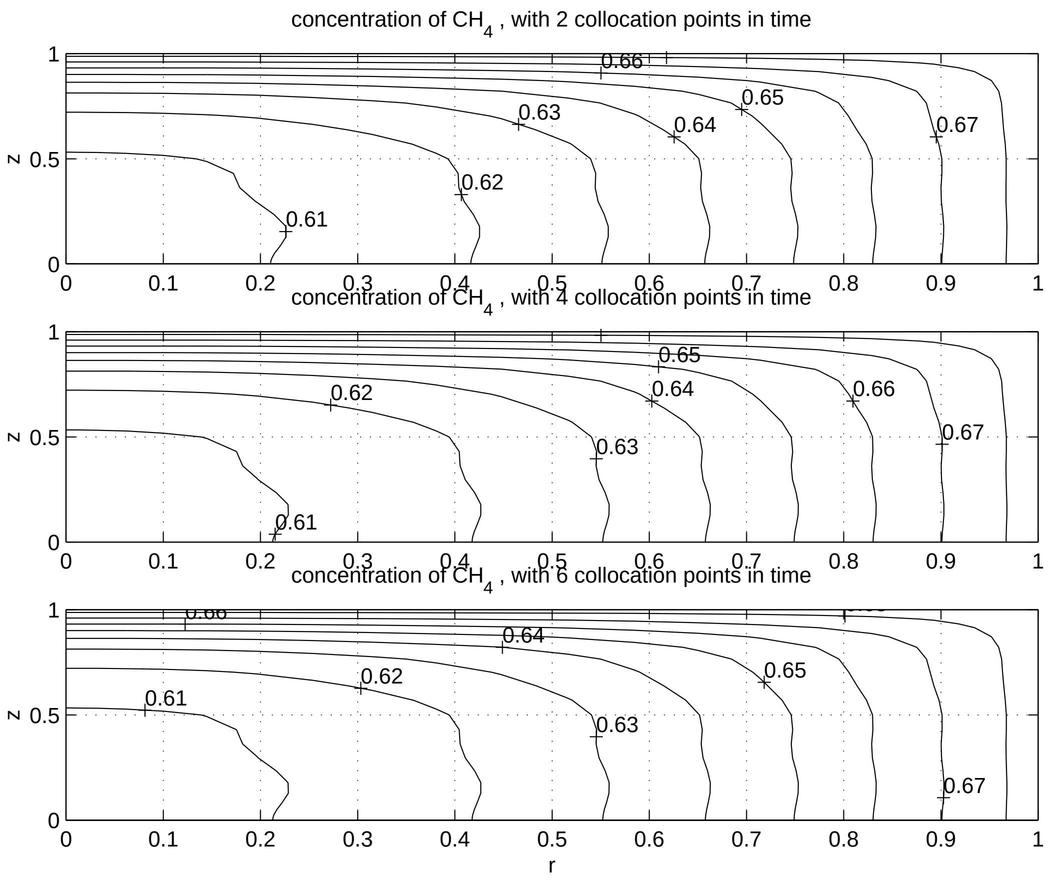

Figure 5.8: Evolution of CH_4 concentration with 2, 4, 6 collocation points in time when odaepc.m is called. Temperature 1400°K and operating pressure 100 Torr. $t = 3$hr.

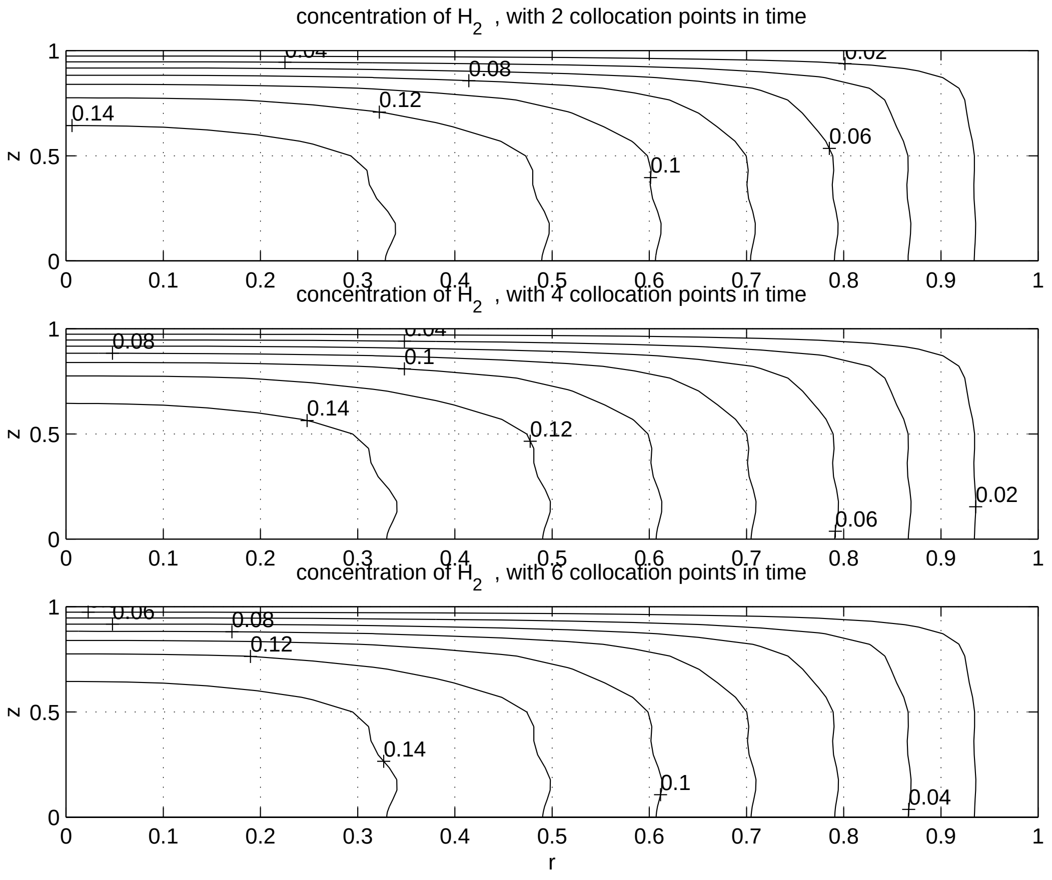

Figure 5.9: Evolution of H_2 concentration with 2, 4, 6 collocation points in time when odaepc.m is called. Temperature 1400°K and operating pressure 100 Torr. $t = 3$hr.

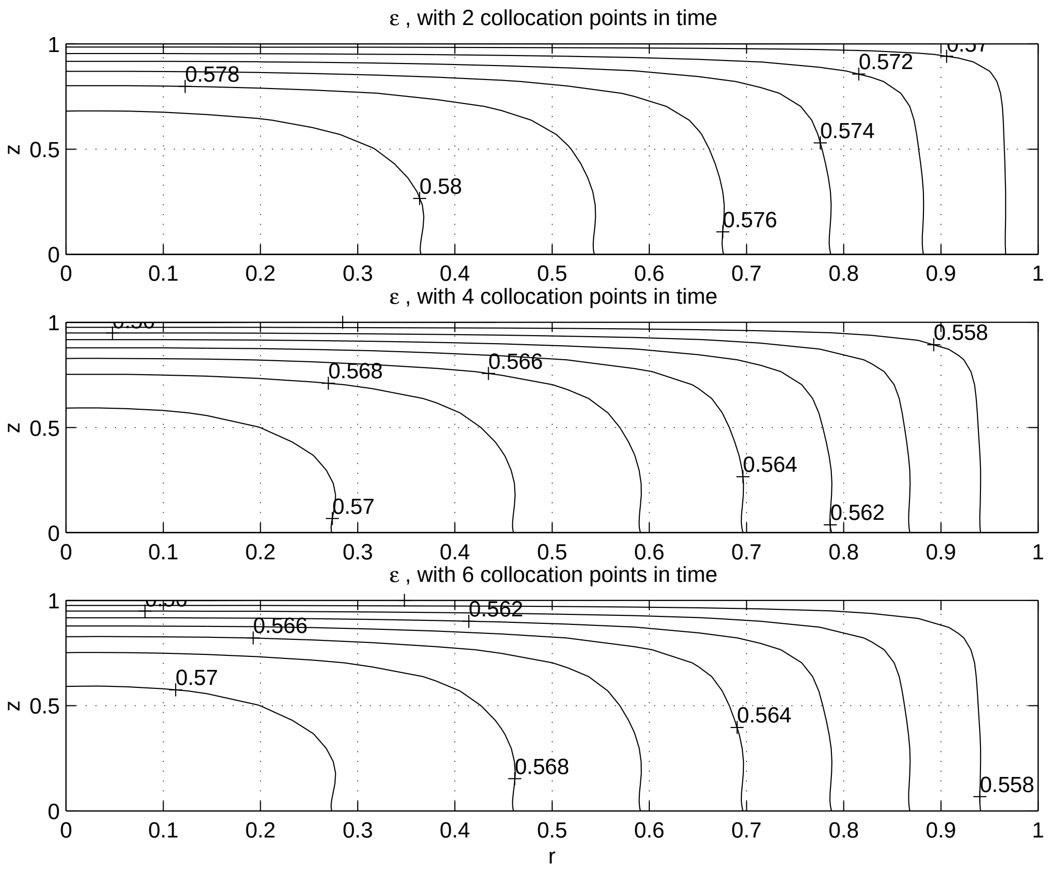

Figure 5.10: Evolution of porosity with 2, 4, 6 collocation points in time when odaepc.m is called. Temperature 1400°K and operating pressure 100 Torr. $t =$ 3hr.

Chapter 6

Conclusions and Recommendations for Further Studies

6.1 Concluding Remarks

An Object-Oriented approach was developed to implement global spectral discretization methods using the Galerkin projection. Data structures for the trial function sets, discretized linear operators, and the scalar fields that result during intermediate MWR computations were developed. The methods were specifically formulated to be applied to 1, 2, 3-dimensional systems.

Time integration using a collocation based integrator was demonstrated. The collocation based integrator was developed to work with the data structures used in the spatial discretization steps.

The applicability of global spectral methods was demonstrated on a two dimensional isothermal carbon-carbon chemical vapor infiltration (CVI) simulation problem. Most of the previous studies by other researchers were based on finite-element formulations.

The convergence of the global spectral method for the CVI problem was studied as a function of both spatial trial function truncation number and time-collocation discretization point number.

6.2 Further Studies

6.2.1 Improvements of the Numerical Technique

The value of this thesis is primary due to the construction of the Object-Oriented programming framework and the software tools to implement high-dimensional simulation and model reduction studies. There are many improvements that can build on this work:

- Computational efficiency: Based on a Sun Sparc ultra-10 workstation, the time integration of the CVI problem with 10×10 trial functions, two collocation points in time will take about 7 minutes to reach $t = 3hr$ simulated time. The time integration of the CVI problem with 6×6 trial functions, six collocation points in time will take about 13 minutes to reach $t = 3hr$ simulated time. Therefore, important progress can be made if some improvements on the design and coding of the computing algorithm reduced the working time.

- Further development in an Object-Oriented approach to organize the physical parameters will increase the software reuse. The goal is to create a framework that would require the least possible changes that needed to be made when the user switches from one simulation problem to a different one.

- Development of Object-Oriented methods for a “heterogeneous” BVP based simulation problems, where the BVPs are defined on separate physical domains.
- Integration of the Object-Oriented based MWR techniques with MATLAB optimization, control, and other toolboxes.

6.2.2 Improvements of the CVI Modeling

In this work, two gaseous species CH_4 and H_2 are applied and a simplified gas diffusion mechanism is used. The transport of the has within the porous medium occurs by a combination of molecular diffusion, Knudsen diffusion, and viscous flow. For dilute gaseous mixtures at relatively high operating pressures, a simple binary diffusion description may suffice. A comprehensive description of the transport processes accounting for the above mechanisms can be obtained from the Dusty Gas model [8, 19, 26].

Suppose there are three gaseous species, in a Dusty Gas model; the gaseous diffusion can be expressed by

$$[\,\tfrac{dc}{dz}\,] = -[F][N_z] \tag{6.1}$$

where $[\,\tfrac{dc}{dz}\,]$ is the concentration gradient and $[N_z]$ is the molar flux.

$$F_{ij} = -\frac{x_i}{D_{ij}}(i \neq j) \tag{6.2}$$

$$F_{ii} = -\frac{1}{D_{iK}} + \sum_{h=1,h\neq i}^{n} \frac{x_h}{D_{ih}} \tag{6.3}$$

where D_{iK} is the Knudsen diffusivity of species i.

For three species example,

$$F = \begin{bmatrix} \frac{1}{D_{1k}(r)} + \frac{x_2}{D_{12}} + \frac{x_3}{D_{13}} & -\frac{x_1}{D_{12}} & -\frac{x_1}{D_{13}} \\ -\frac{x_2}{D_{21}} & \frac{1}{D_{2k}(r)} + \frac{x_1}{D_{21}} + \frac{x_3}{D_{23}} & -\frac{x_2}{D_{23}} \\ -\frac{x_3}{D_{31}} & -\frac{x_3}{D_{32}} & \frac{1}{D_{3k}(r)} + \frac{x_1}{D_{31}} + \frac{x_2}{D_{32}} \end{bmatrix} \tag{6.4}$$

One of the drawbacks of the isothermal CVI process is it's long reaction time. This can be overcome by using a thermal-gradient system. In a thermal-gradient CVI process, the highest temperature is at the center of the preform, or at one end, depending on the heating mechanism. The reactant gas diffuses through the preform from the cold to the hot surface. This allows the reactant gases to diffuse to the hot end before reacting to deposit the matrix at the hot end. If the gradients are large enough, a moving densification front is created so that dense matrix grows from the hot end to the cool end of the preform. A higher deposition rate than for isothermal processes can be achieved without the 'canning' problem with such a process. To model a CVI process with thermal gradients, a temperature governing equation must be added.

BIBLIOGRAPHY

[1] Adomaitis, R. A. and Lin, Y. -h. 2000, A collocation/quadrature-based Sturm-Liouville solver. *Appl. Math. Comp.*, **110**, 205-223.

[2] Adomaitis, R. A., Lin, Y. -h. and Chang, H. -Y. 2000, A Computational Framework for Boundary-Value Problem Based Simulations. *Simulation*, **74**, 28-38.

[3] Bammidipati, S., Stewart, G. D., etc. 1996. Chemical vapor deposition of carbon on graphite by methane pyrolysis. *AIChE J.*, **42(11)**, 3123-3132.

[4] Bird, R. B., Stewart, W. E. and Lightfoot, E. N. 1960, *Transpot phenomena*, New York: John Wiley & Sons.

[5] Buckley, J. D. 1988, Carbon-carbon, an overview. *Ceram. Bull.*, **67(2)**, 364-368.

[6] Chung, G. -Y. and McCoy, B. 1991, Modeling of chemcal vapor infiltration for ceramic composites reinforced with layered, woven fabrics. *J. Am. Ceram. Soc.* , **74(4)**, 746-751.

[7] Deepak and Evans, J. W. 1993, Mathematical model for chemical vapor infiltration in a microwave-heated preform. *J. AM. Ceram. Soc.* , **76(8)**, 1924-1929.

[8] Feng, C. and Stewart, W. E. 1973, Practical models for isothermal diffusion and flow of gases in porous solids. *Ind. Eng. Chem. Fundam.*, **12(2)**, 143-147.

[9] Fletcher, C. A. J. 1984, *Computational Galerkin methods*, New York: Springer-Verlag.

[10] Funaro, D. 1992, *Polynomial approximation of differential equations*, New York: Springer-Verlag.

[11] Gottlier, D. and Orszag, S. A. 1977. *Numerical analysis of spectral methods: theory and applications*, CBMS-NSF regional conference series.

[12] Guote, S. M. and Tsamopoulos,J. A. 1990, Forced-flow chemical vapor infiltration of porous ceramic materials. *J. Electrochem. Soc.*, **137(11)**, 3675-3681.

[13] Lin, Y. H., Chang, H. Y. and Adomaitis, R. A. 1999, MWRtools: a library for weighted residual method calculations. *Computers and Chemical Engineering*, **23**, 1041-1061.

[14] Lin, Y. S. and Burggraaf, A. J. 1991, Modeling and analysis of CVD processes in porous media for ceramic composite preparation. *che. Eng. Sci.*, **46(12)**, 3067-3076.

[15] Mason, E. A. and Evans, R. B. 1969, Graham's laws: simple demonstrations of gases in motion. *J. Chem. Edu.*, **46(6)**, 358-364.

[16] McAllister, P. and Wolf, E. E. 1990, Modeling of chemical vapor infiltration of carbon in porous carbon substrates. *Carbon*, **29(3)**, 387-395.

[17] McAllister, P. and Wolf, E. E. 1993, Simulation of a multiple substrate reactor for chemical vapor infiltration of pyrolytic carbon within carbon-carbon composites. *AIChe Journal*, **39(7)**, 1196-1209.

[18] Middleman, S. 1989, The interaction of chemical kinetics and diffusion in the dynamics of chemical vapor infiltration. *J. Mater. res.* , **4(6)**, 1515-1524.

[19] Midha, V. and Economou, D. J. 1997, A two-dimensional model of chemical vapor infiltration with radio-frequency heating. *J. Electrochem. Soc.*, **144(11)**, 4062-4071.

[20] Midha, V. and Economou, D. J. 1998, A two-dimensional model of chemical vapor infiltration with radio-frequency heating. *J. Electrochem. Soc.*, **145(10)**, 3569-3580.

[21] Morell, J. I., Economou, D. J. and Amundson, N. R. 1992, A mathematical model for chemical vapor infiltration with volume heating. *J. Electrochem. Soc.*, **139(1)**, 328-336.

[22] Ofori, J. Y. and Sotirchos, S. V. 1997, Multidimensional modeling of chemical vapor infiltration: application to isobaric CVI. *Ind. Eng. Chem. Res.*, **36-2**, 357-367.

[23] Savage, G. 1993, *Carbon-carbon composites*, London: Chapman & Hall.

[24] Skamser, D. J., Jennings, H. M. and Johnson, D. L. 1997, Model of chemical vapor infiltration using temperature gradients. *J. Mater. Res.*,**12(3)**, 724-737.

[25] Tomadakis, M. M. and Sotirchos, S. V. 1991, Effective Knudsen diffusivities and properties of structures of unidirectional fibers. *AIChE J.*, **37(8)**, 1175-1186.

[26] Tomadakis, M. M. and Sotirchos, S. V. 1991, Knudsen diffusivities in strutures of randomly overlapping fibers. *AIChE J.*, **37(1)**, 74-85.

[27] Vaidyaraman, S., Lackey, W. J., Agrawal, P. K. and Starr, T. L. 1996, 1-D model for forced flow-thermal gradient chemical vapor infiltration process for carbon/carbon composites. *Carbon*, **34(9)**, 1123-1133.

[28] Vaidyaraman, S., Lackey, W. J., Freeman, G. B., Agrawal, P. K. and Langman, M. D. 1995, Fabrication of carbon-carbon composites by forced flow-thermal gradient chemical vapor infiltration. *J. Mater. Res.*, **10(6)**, 1469-1476.

[29] Villadsen, J. V. and Stewart, W. E. 1967, Solution of boundary-value problems by orthogonal collocation. *Chem. Engng. Sci.*, **22**, 1483-1501.