University of Alberta

Experimental testing and numerical simulation of interfacial coupling phenomena in two-phase flow through porous media

by

Xiao Yun Zhang ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the

requirements for the degree of Master of Science

in

Petroleum Engineering

Department of Civil and Environmental Engineering

Edmonton, Alberta, Canada

Spring 2006

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-13921-8
Our file Notre référence
ISBN: 0-494-13921-8

Dedication

To my family,

for being always supportive

Abstract

Interfacial coupling phenomena in two-phase flow through porous media was tested using experimental and numerical simulation techniques.

Through the analysis of the experimental and numerical simulation results, it was found that the Modified Transport Equations model, which took into account interfacial coupling effects, gave a better description of two-phase cocurrent flow through porous media than did the traditional transport equations model. Interfacial coupling effects make the flood front steeper, and hence delay water breakthrough. Moreover, for a given water flooding reservoir, interfacial coupling effects decrease as the injection rate increases.

Based on a sensitivity analysis of the numerical simulation results, it was found that (a) the capillary coupling effect is more significant than the hydrodynamic and viscous coupling effects; (b) for a given sand-fluid system, the higher the reservoir angle of dip, the larger the breakthrough time, and the smaller the influence of interfacial coupling effects.

Acknowledgement

I would like to express my thanks to my supervisors, Dr. Ramon G. Bentsen and Dr. Luciane B. Cunha, for their kind supervision, guidance and encouragement throughout the period of this study. I appreciate both their efforts in reading through the manuscripts and making necessary corrections.

It is also important to thank Mr. Sean Watt for his help during the conduct of the experiments. Thanks are also due to Mr. Roy Gitzel for upgrading the data acquisition system.

Last, but not least, I thank the other faculty members of the School of Mining and Petroleum Engineering of the University of Alberta. In particular, I thank Dr. Ergun Kuru, Dr. Marcel Polikar, Dr. Tayfun Babadagli, and Dr. Peter Toma for their valuable courses and their words of encouragement and advice. Financial support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

Table of contents

List of tables

List of figures

Nomenclature

Roman letters

A_c	area under the capillary pressure curve, M/(T^2L)
A_o and B_o	fitting coefficients
a^*	coefficient to be determined experimentally
a_i, b_i, c_i, d_i	i=null, 0, 1, 2. fitting coefficients
$C(S)$	capillary term; $C(S) = -\frac{1}{M_r R_{12}(S)} \lambda(S) F_1(S) k_{r2}(S) \frac{d\pi_c}{dS}$
f	wetting phase fractional flow
f_o	the frequency response, 1/T
g	acceleration due to gravity, L/T^2
$G(S)$	gravity term; $G(S) = \left(1 - \frac{\lambda(S)\{1 + (\rho_2/\Delta\rho')[1 - R_{12}(S)]\} N_g k_{r2}(S)}{R_{12}(S) M_r} \right) F_1(S)$
GUI	graphical user interface
h	thickness of the core, or relative height, L
J	the coefficients of the Jacobian matrix
$JAVA^{TM}$	trademark of Sun Microsystems Inc.
k	reference permeability = absolute permeability, k_{1r} or k_{2r}, L^2
k_r	relative permeability
k_{1r}	permeability to wetting phase at residual non-wetting phase saturation, L^2
k_{2r}	permeability to non-wetting phase at initial or irreducible wetting phase saturation, L^2
k_{r1}	relative permeability to the wetting phase
k_{r2}	relative permeability to the nonwetting phase
k_{r11} and k_{r22}	the relative permeability of each fluid phase if it were alone
k_{r12} and k_{r21}	the coupling effect which arises due to the introduction of one more fluid into the system
l	the characteristic dimension of the pores of a core
L	length of the core, L
LabVIEW	trademark of National Instruments Corporation

LHS	left hand side of an equation
M	the morphology of the pores
M_r	end point mobility ratio; $M_r = \frac{k_{1r}}{\mu_1} \cdot \frac{\mu_2}{k_{2r}} = \frac{\lambda_{1r}^0}{\lambda_{2r}^0}$
N_c	capillary number; $N_c = \frac{A_c \lambda_{1r}^0}{vL}$
N_g	gravity number; $N_g = \frac{\lambda_{1r}^0 \Delta\rho' g \sin(\theta)}{v}$
P	phase pressure, M/(T^{2}L)
P_c	capillary pressure, M/(T^{2}L)
P_d'	displacement pressure, M/(T^{2}L)
RHS	right hand side of an equation
R_{12}	hydrodynamic factor; $R_{12} = 1 - a(1 - S)$
S	normalized wetting phase saturation; $S = \frac{S_1 - S_{1i}}{1 - S_{2r} - S_{1i}}$
S_1	wetting phase saturation in the core
S^*	normalized variable inlet saturation
S_{1i}	initial or irreducible wetting phase saturation
S_{2r}	residual non-wetting phase saturation
SSCO	steady-state cocurrent
SSCT	steady-state countercurrent
t	time, T
USCO	unsteady-state cocurrent
USCT	unsteady-state countercurrent
v	phase flux, L/T
x	distance along the length of the core, L

Greek letters

α_{ci}	the capillary coupling parameters.
α_i	interfacial coupling factor; $\alpha_i = \alpha_{vi}\alpha_{ci}$, i=1, 2
α_{ij}	i, j =1, 2. partition coefficients

α_{vi}	the viscous coupling parameters
β	the wetting angle
Δf	the vector of unknowns
ΔS	saturation grid size
$\Delta \tau$	time step size
ΔP	pressure gradient, M/(T^{2}L^2)
$\Delta\rho'$	difference in density without hydrodynamic effect; $\Delta\rho' = \rho_1 - \rho_2$, M/L^3
ε, ε_1, ε_2	predefined small value
ξ	normalized distance from the inlet surface of a core; $\xi = \dfrac{x}{L}$
θ	reservoir or core angle of dip
λ	mobility, TL^3/M
λ_i^0	i =1,2, the mobilities measured in a steady state cocurrent (SSCO) flow experiment, TL^3/M
λ_i^*	i =1,2, the mobilities measured in a steady state countercurrent (SSCT) flow experiment, TL^3/M
λ_{ij}	generalized mobility of phase i, j =1, 2, TL^3/M
λ_m^0	parameter to ensure dimensional consistency, TL^3/M
λ_{1r}^0	effective mobility to wetting phase at residual saturation of non-wetting phase, TL^3/M
λ_{2r}^0	effective mobility to non-wetting phase at initial saturation of wetting phase, TL^3/M
ƛ(S)	average coupling coefficient; ƛ$(S) = \dfrac{\alpha_1 + \alpha_2}{2}$
$\pi_c(S)$	dimensionless capillary pressure; $\pi_c(S) = \dfrac{P_c - P_d'}{A_c}$
ρ	density of fluids, M/L^3
τ	normalized time or measured injected normalized pore volume; $\tau = \dfrac{vt}{\phi L(1 - S_{2r} - S_{1i})}$
$\tilde{\tau}$	calculated injected normalized pore volume

ϕ	porosity of the porous medium
ψ	potential, M/(T^{2}L)
μ	viscosity, M/(TL)
σ	interfacial tension, M/T^2

Superscripts

0	denotes the parameter is for cocurrent flow
*	denotes the parameter is for countercurrent flow

Subscripts

1	wetting phase
2	non-wetting phase
i	irreducible
r	residual

The equations and the calculated results in this document are presented in a consistent set of units; the units of variables and parameters are expressed in length (L), *mass* (M) *and time* (T).

CHAPTER 1

Introduction

1.1 Background

In terms of the development of petroleum reservoirs, it is extremely important to forecast accurately and optimize future production performance, and to determine the ultimate recovery factor. To achieve this purpose, one has to understand fully the physics underlying the reservoir process, multiphase flow through porous media. In this regard, the transport equations are the fundamental issue to be resolved.

Nowadays, the widely used transport equations of multiphase flow through porous media are Muskat's extension of Darcy's equations [Muskat and Meres (1936), Muskat et al. (1937)]. In these equations, the interactions between the simultaneously flowing phases are neglected and Darcy's equation, an empirical equation for single phase flow through porous media, is assumed to be independently applicable, without any cross effects, to each phase during multiphase flow through porous media.

However, as pointed out by several researchers [Muccino et al. (1998), Bentsen (1998a, 1998b), Ayub and Bentsen (1999), Rose (1999, 2000)], Muskat's extension of Darcy's equation can not give accurate recovery predictions when used to simulate multiphase flow in petroleum reservoirs. Moreover, experimental data [Lelièvre (1966), Bourbiaux and Kalaydjian (1990), Bentsen and Manai (1991, 1993)] have shown that the magnitude of the relative permeability for a given phase, obtained from countercurrent flow at a given saturation, is always less than that acquired from a cocurrent experiment conducted in the same porous medium. This phenomenon is also encountered in industrial practice, such as the SAGD (steam assisted gravity drainage) process [Nasr et al. (2000)]. Many researchers suspected that the reason for this is that the presence of one fluid affects the flow of the other fluids, leading to interfacial coupling effects which should be, but are not, taken into account in Muskat's extension of Darcy's equation.

How to account for the interactions between the simultaneously flowing phases in the porous medium, or incorporate the interfacial coupling effects into the transport equations, has been a problem for a long while, and has been intensively studied over the last two decades. de la Cruz and Spanos (1983), Whitaker (1986) and Kalaydjian (1987, 1990) applied volume averaging techniques on the pore scale Navier–Stokes equations to obtain a set of sophisticated transport equations, on the macroscopic scale, for one dimensional two phase flow. This set of equations has four transport coefficients, two of which represent the interaction between the two fluids flowing in the same porous medium, and, hence, can explain the discrepancy between the relative permeabilities measured during countercurrent flow and those obtained during cocurrent flow. The shortcomings of these four coefficient transport equations (referred to, in this study, as the Four Coefficient Transport Equations), as pointed out by Bentsen (2001), are that some information is lost when the volume averaging technique is used to pass from the microscopic to the macroscopic scale. Hence, the source and magnitude of interfacial coupling is not well understood. Moreover, it is hard to determine the transport coefficients accurately by using currently available techniques and equipment.

Trying to overcome the shortcomings associated with the Four Coefficient Transport Equations mentioned above, Bentsen (2001, 2003) constructed, by introducing a partitioning concept into the Four Coefficient Transport Equations, a modified set of transport equations (referred to, in this study, as Modified Transport Equations) for one dimensional two phase flow. From these Modified Transport Equations, one can see easily the significance of the different driving forces which give rise to the flux, and identify the source and magnitude of the interfacial coupling. In addition, these Modified Transport Equations contain two instead of four transport coefficients which can be determined easily using one set of traditional, horizontal, two phase, steady state cocurrent (SSCO) flow experiments. Now, the work left to be done is to validate experimentally these Modified Transport Equations.

1.2 Purpose of the study

As is known, no theoretical work can establish its credibility unless and until validated by laboratory experimental work, especially in applied scientific disciplines. To validate the Modified Transport Equations, Ayub (2000) and Ayodele (2004) have done a lot of work. This study is undertaken to provide further testing of the applicability of the Modified Transport Equations to cocurrent flow situations, and to find out the existence and significance of the interfacial coupling effects.

1.3 Methodology

Whether or not the mathematical model of a phenomenon catches the underlying physics really depends on how well the solutions of the model match corresponding experimental results. Therefore, to achieve the goals set out in the previous section, it was decided to solve the Modified Transport Equations by developing a numerical simulator and to compare these solutions with experimental data acquired in the laboratory and those found in the literature. The significance of the interfacial coupling effects under various conditions will be determined by performing sensitivity analyses.

1.4 Report structure

Chapter 2 reviews the literature. Chapter 3 goes through the derivation of the Modified Transport Equations and the definitions of interfacial parameters, and gives the related validation methods. Chapter 4 deals with the experimental issues, and Chapter 5 contains the development and the validation of the numerical simulator. In Chapter 6, the match of experimental and simulation results is given, and the sensitivity analyses are carried out as well. Chapter 7 provides the conclusions and recommendations.

CHAPTER 2

Literature review

To study a complicated phenomenon, it is usual to start with a simplified case to gain some insight and then to generalize it to a more complicated case. In the studies of the phenomena of multiphase flow through porous media, the same logic is followed. All the discussions and analyses in this research are confined to the simplified case of multiphase flow through porous media which is the one dimensional flow of two immiscible and incompressible fluids through a homogenous and isotropic porous medium.

2.1 Approaches to studying two-phase flow in porous media

To describe the macroscopic behavior of two-phase flow in porous media, in addition to the method of generalizing the Darcy's law, which was used by Muskat and Meres (1936) and Muskat et al. (1937) to get the well known Muskat's extension of Darcy's equation, several other research approaches, such as (1) the volume averaging methods, (2) the principles of irreversible thermodynamics, (3) the use of analogous models and (4) the continuum theory of mixtures, have been developed. Ayub (2000) presented a detailed literature review of these four newly developed approaches and found out that all of them result in transport equations having a similar appearance; that is, instead of two transport coefficients as shown in Muskat's extension of Darcy's equation, all of these new transport equations have four transport coefficients, two of which represent the interaction between the two fluids flowing in the same porous medium. Due to the similarity among the transport equations with four transport coefficients, only the equations proposed by de la Cruz and Spanos (1983), Whitaker (1986) and Kalaydjian (1987, 1990) are discussed here, and compared to Muskat's extension of Darcy's equation.

2.2 Muskat's extension of Darcy's equation

2.2.1 Darcy's law

As is well known, Darcy's law, which is based on several assumptions, provides a rigorous description of single phase flow of a fluid through porous media under the effect of gravity and viscosity forces. Darcy's law can be formulated as follows:

$$v = -\frac{k}{\mu}(\nabla P - \rho g \sin\theta)\,. \tag{2.1}$$

The definitions of all terms in the above equation can be found in the Nomenclature. From this point on, the definitions of the symbols in all the equations presented in this study are given in the Nomenclature, even if they are explained in the text.

As pointed out by Rose (1999), the assumptions for Darcy's law are: (a) the fluid saturating the pore space can be regarded as a homogeneous and incompressible Newtonian liquid that completely fills the conducting network of interconnected pathways; (b) the process under study is a steady-state, isothermal, and a limiting one that is occurring under low Reynolds Number conditions; (c) the porous medium locally is uniform, isotropic, chemically inert, and non-deformable; and (d) at the pore level (that is, microscopic) frame of reference, a zero velocity no-slip boundary condition can be presumed to exist at all stationary interstitial fluid/pore surfaces.

To test the applicability of Darcy's law to two-phase flow problems, several experimental studies were conducted using two immiscible fluids [Cloud (1930), Plumer et al. (1937), and Fletcher (1949)]. These experimental studies found that the presence of a second phase can cause a reduction in the permeability to the first phase and decrease the permeability of the mixture. The must probable reason for this phenomenon is the violation of assumption (d) in that fluid/fluid interface is created in the two-phase flow case.

2.2.2 Extending Darcy's law to two phase flow

Without satisfactorily explaining the effect of fluid/fluid interface on the permeability in the two-phase problem, Muskat and his colleagues [Muskat and Meres (1936); Muskat et al. (1937)] assumed that Darcy's law should be valid for each fluid phase of the two simultaneously flowing fluids in the same porous medium and introduced terms like relative permeability (k_{r1} and k_{r2}) into Darcy' equation to account for the reduction in the permeability due the presence of a second fluid. They finally obtained the following formulations for two phase flow in porous media:

$$v_1 = -\frac{kk_{r1}}{\mu_1}(\nabla P_1 - \rho_1 g \sin\theta) \qquad (2.2)$$

and

$$v_2 = -\frac{kk_{r2}}{\mu_2}(\nabla P_2 - \rho_2 g \sin\theta)\,. \qquad (2.3)$$

These equations are usually called Muskat's extension of Darcy's equations. The phase pressures are connected by the capillary pressure equation, which was proposed firstly by Leverett (1941), as shown below:

$$P_c = \frac{\sigma \cos(\beta)}{\sqrt{k/\phi}} \mathit{function}(M, S_1) = P_2 - P_1, \qquad (2.4)$$

where, M is the morphology of the pores.

It needs to be noted that capillary pressure is subject to hysteresis, as the wetting angle β is a function of the direction of the displacement. One has to simulate the same rock-fluids system, and displacement process, that was used to determine experimentally the capillary pressure.

By combining Muskat's extension of Darcy's equation with the continuity equation (the equation of mass conservation), equations of state and some boundary and initial conditions, it is possible to solve the problems involved in two immiscible, incompressible fluids flowing through a homogenous and isotropic porous medium, if the assumptions underlining the Muskat's extension of Darcy's equation and relative permeability are acceptable, which, however, seems not to be true.

2.2.3 Problems associated with Muskat's extension of Darcy's equation

2.2.3.1 Improper assumptions

To use Muskat's extension of Darcy's equation, one has to accept the assumption that Darcy's law is valid for each fluid phase simultaneously flowing through the same porous medium. This assumption implies that the boundary conditions at the fluid/fluid and fluid/solid interfaces are the same, or at least similar, which means that the velocity of the flowing fluids at the fluid/fluid interface is zero, or almost zero; however, on the basis of viscous flow theory, this is not true. In addition, by using Poiseuille-type concentric flow in a circular tube as a model representing the simultaneous flow of two immiscible fluids in a porous medium, Yuster (1951) found that a finite velocity must be taken as the boundary condition at the fluid-fluid interfaces.

2.2.3.2 Factors affecting relative permeability

As mentioned above, the underlying theory of Muskat's extension of Darcy's equation depends on the concept of relative permeability; therefore, it is important to explore this concept in detail. Marle (1981) studied relative permeability by using dimensional analysis and found that it should be a function of several dimensionless groups as follows:

$$k_{r1} \text{ or } k_{r2} = \text{function of} \left(\frac{\rho_1}{\rho_2}, \frac{(\rho_1 - \rho_2)gl}{\sigma}, \frac{\rho_1 l v}{\mu_1}, \frac{\mu_1 v}{\sigma}, \frac{\mu_1}{\mu_2}, \beta, S_1 \right),$$

where

$\frac{(\rho_1 - \rho_2)gl}{\sigma}$ = the ratio of the force of gravity to the capillary forces,

$\frac{\rho_1 l v}{\mu_1}$ = the Reynolds number, the relationship of the forces of inertia to the forces of viscosity,

$\frac{\mu_1 v}{\sigma}$ = the ratio of the forces of viscosity to the capillary forces, and

β = the wetting angle.

Among these dimensionless groups, Marle (1981) pointed out that the effect on relative permeability of the first three groups is fairly small and can be neglected, while the last four, especially the last two groups, have much more influence. While the effect of wetting phase saturation is widely accepted, and the influence of velocity on relative permeability is debatable, the influence of interfacial tension, viscosity ratio and the wetting angle are supported by the following facts.

Firstly, the importance of interfacial tension has been demonstrated in the water flooding process. Lowering the interfacial tension causes an increase in the proportion of oil recovered and a decrease in the value of oil saturation, for which its relative permeability becomes zero [Marle (1981)].

Secondly, the impact of the viscosity ratio on relative permeability is demonstrated by the fact that the sum $k_{r1} + k_{r2}$ is less than unity [Bear (1972)]. Another example to show that the viscosity ratio should affect k_{r1} and k_{r2} is that the relative permeability to oil of a reservoir containing small amounts of connate water in pendular saturation might be greater than one [Bear (1972)]. All of these phenomena are called viscous coupling which refers to the coupling that arises due to the viscous drag exerted by one fluid on the other when they flow through the same porous medium, and it is usually associated with the mobility of the fluids.

Finally, the influence of wettability, indicated by the wetting angle which is subject to hysteresis, on relative permeability has been widely known in that the relative permeability obtained from the imbibition process is different from that acquired from the drainage process.

In summary, relative permeability, instead of depending only on the saturation, as assumed in Muskat's extension of Darcy's equation, is also a function of at least three other variables: σ, β, and μ_1/μ_2. It seems that σ and β will not cause too much difference if one uses the relative permeability measured from a certain rock-fluids system and process (imbibition/drainage) to simulate the same system and process. Nevertheless, how to deal with μ_1/μ_2 is a problematic issue for Muskat's extension of Darcy's equation and hence an attempt at quantifying the influence of μ_1/μ_2 on two phase flow and incorporating it into the transport equations has been underway for some time.

2.2.3.3 Difficulties in explaining experimental results

Experimental data [Lelièvre (1966), Bourbiaux and Kalaydjian (1990), Bentsen and Manai (1991, 1993)] have shown that the magnitude of the relative permeability for a given phase at a given saturation obtained from countercurrent flow is always less than that acquired from a cocurrent experiment conducted in the same porous medium, which can not be explained by the normal transport equations, Muskat's extension of Darcy's equations. People thought that this problem can be attributed to Muskat's extension of Darcy's equation, because Muskat's extension does not take into account the coupling effect that arises because of the introduction of one more fluid. This coupling effect was originally thought to be viscous coupling; that is, coupling due to momentum transfer, or viscous drag, between two simultaneously flowing fluids in the same porous medium [Ayub and Bentsen (1999)]. However, it has been found that the viscous coupling effect is not large enough to make such a big discrepancy between the relative permeabilities measured in cocurrent and countercurrent flow experiments [Zarcone and Lenormand (1994), Bentsen (2003)]. There exists the possibility that there might be an additional source of interfacial coupling, namely the coupling of pressure that takes place across the interfaces of the fluids located in a porous medium [Babchin and Yuan (1997), Bentsen (2001)]. Such coupling is referred to as capillary coupling [Bentsen (2001)]. Bentsen (2001) also noted that the effects attributable to viscous coupling must be associated with the mobilities of the flowing phases, while the effects attributable to capillary coupling must be associated with the potential gradients acting across the flowing phases. To address the problem that Muskat's extension of Darcy's equation does not account for interfacial coupling effects during the two-phase flow process in porous media, researchers have constructed more sophisticated transport equations.

2.3 Four Coefficients Transport Equations

It is known that the physical laws governing equilibrium and the flow of two fluids in a porous medium, at the pore level, are not that difficult. If both fluids are Newtonian fluids and they are flowing in the laminar region under isothermal conditions, which is the usual case in the development of a conventional petroleum reservoir, the flowing process at the pore level can be described mathematically by the Navier-Stokes equations, and the boundary conditions at the fluid-rock and fluid-fluid interfaces can be well defined also. However, due to the complexity of the pore geometry, the macroscopic behavior of two-phase flow, with which one is concerned in engineering practice, is not so easily deduced from pore level behavior.

Starting from the Navier-Stokes equations at the microscopic level, and by considering the behavior of the two flowing phases to be random in nature and by using volume averaging techniques, de la Cruz and Spanos (1983), Whitaker (1986) and Kalaydjian (1987, 1990) arrived at a macroscopic mathematical description of two-phase flow through porous media:

$$v_1 = -\frac{kk_{r11}}{\mu_1}\frac{\partial \psi_1}{\partial x} - \frac{kk_{r12}}{\mu_2}\frac{\partial \psi_2}{\partial x} \qquad (2.5)$$

and

$$v_2 = -\frac{kk_{r21}}{\mu_1}\frac{\partial \psi_1}{\partial x} - \frac{kk_{r22}}{\mu_2}\frac{\partial \psi_2}{\partial x}, \tag{2.6}$$

where k_{r11} and k_{r22} represent the relative permeability of each fluid phase if it were alone, while k_{r12} and k_{r21} represent the coupling effect which arises due to the introduction of one more fluid into the system.

Compared with Muskat's extension of Darcy's equation, the flux for a given phase in the new equations is not only proportional to one driving force, the pressure gradient acting across the phase, but also proportional to the pressure gradient of the other phase through a cross, or coupling term which is supposed to account for the coupling effect. At the time the new equations were derived, the coupling effect was thought to be due to viscous coupling. However, due to the general form of the new equations, and the way these equations were derived, the cross term might include both viscous and capillary coupling.

By analyzing the relationships between Muskat's extension of Darcy's equation and the Four Coefficients Transport Equations, Kalaydjian (1990) provided an experimental way to determine the mobility matrix. The determination is based on two experiments, the first one a horizontal, steady state, cocurrent displacement without any capillary effect, and the second one a vertical, steady state, countercurrent flow experiment which has a zero total flow rate. With the data obtained from these two experiments, Kalaydjian (1990) found that application of the newly derived four transport coefficient equations to flow in porous media gave an explanation for the discrepancies measured between standard cocurrent and countercurrent mobilities.

2.3.1 Determination of the transport coefficients

Due to the difficulties associated with measuring the gravitational effect when conducting vertical, steady state, countercurrent flow experiments as proposed by Kalaydjian (1990), Bentsen and Manai (1991, 1993) used horizontal, steady state, cocurrent and countercurrent experiments to obtain the transport coefficients of the

Four Coefficients Transport Equations. The experimental difficulties encountered, especially in the practical case where the experiments have to be conducted on a tight core, were the selection of the selectively wet material which had to have a high threshold pressure to prevent the oil phase flowing through, and a high permeability to allow the water phase to flow through easily.

Liang and Lohrenz (1994) proposed a combination of steady-state and unsteady-state experiments to measure the transport coefficients. This should be a good approach to determine the coefficients provided that the phase pressure can be measured accurately, and provided that there are sufficient pressure measurement points to construct a fairly good pressure gradient profile.

In addition to the approaches mentioned above, several analytical and numerical solutions [Liang (1993), Bear and Bachmat (1991), Rose (1990)] have been developed, under certain idealized conditions, to calculate the coefficients, but due to the complexity of the effect factors, applicable analytical or numerical solutions without any restrictions have not been possible.

2.3.2 The problems associated with Four Coefficients Transport Equations

There are two major problems associated with the Four Coefficients Transport Equations which are, (a) it is difficult to measure the transport coefficients accurately; (b) the source and magnitude of the interfacial coupling is not well understood.

In terms of the operational issue, difficulty (a) can be easily understood based on the discussion in Section 2.3.1. Even if one uses the approach proposed by Liang and Lohrenz (1994), the easier experiment to conduct compared with the others mentioned above, the problematic phase pressure measurement and the difficulty in relating the pressure gradient to a certain saturation point on the rapidly changing saturation profile will not give too much credibility to the calculated results.

Problem (b) arises during the derivation of the equations. As pointed out by Bentsen (2001), when the volume averaging approach is used, the mobilities that appear in the transport equations are defined in terms of surface integrals. Because of the complicated nature of the pore space in natural porous media, one needs to be aware that the information lost in passing from the microscopic to the macroscopic scale may not be recaptured easily in the experiment used to determine the mobilities.

2.4 Modification of Four Coefficients Transport Equations

After realizing the shortcomings of the Four Coefficients Transport Equations, Bentsen (2001, 2003) introduced a partition concept into these equations and constructed a set of modified transport equations that enable a better understanding of the role of interfacial coupling in two phase flow through porous media. The formulations of the one dimensional form of the Modified Transport Equations for cocurrent flow are:

$$v_1 = -\lambda_1^0\left\{\frac{\partial\psi_1}{\partial x}+\left(\frac{1-\alpha_1}{2}\right)\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta\right]\right\} \tag{2.7}$$

and

$$v_2 = -\lambda_2^0\left\{\frac{\partial\psi_2}{\partial x}-\left(\frac{1-\alpha_2}{2R_{12}}\right)\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta\right]\right\}. \tag{2.8}$$

For countercurrent flow, these equations are:

$$v_1^* = -\alpha_1\lambda_1^0\left\{\frac{\partial\psi_1}{\partial x}+\left(\frac{1-\alpha_1}{2}\right)\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta\right]\right\} \tag{2.9}$$

and

$$v_2^* = -\alpha_2\lambda_2^0\left\{\frac{\partial\psi_2}{\partial x}-\left(\frac{1-\alpha_2}{2R_{12}}\right)\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta\right]\right\}. \tag{2.10}$$

The coupling parameters, α_1 and α_2, are defined in the paper by Bentsen (2003) and in the thesis by Ayodele (2004) and their defining equations are also briefly presented in

Chapter 3 for the reader's convenience. The hydrodynamic factor can be determined by using a SSCO flow experiment, or just be neglected. The neglect of the hydrodynamic factor only introduces a relative error of about 1% [Bentsen (1998a and 1998b)].

As compared with the Four Coefficients Transport Equations proposed by de la Cruz and Spanos (1983), Whitaker (1986) and Kalaydjian (1987, 1990), the biggest advantage of using these Modified Transport Equations is that people can easily simulate the process of both cocurrent and countercurrent two phase flow in porous media by just measuring two coefficients, λ_1^0 and λ_2^0, as is usually done when using the traditional transport equations, Muskat's extension of Darcy's equations.

However, as mentioned before, any theoretical work has to be supported or validated experimentally before it can be accepted. This study tries to provide some further validation of these Modified Transport Equations.

CHAPTER 3

Theory and related validation methods

Before proceeding further in this study, the basic theory is reviewed. Moreover, the derivation of the Modified Transport Equations, and the definitions of the interfacial coupling parameters are provided. The derivation procedures presented follow closely those provided by Bentsen (2003) and Ayodele (2004). The definitions of the interfacial coupling parameters are taken from the paper by Bentsen (2003) and from the thesis by Ayodele (2004).

3.1 Basic equations

The Four Coefficients Transport Equations [de la Cruz and Spanos (1983), Whitaker (1986) and Kalaydjian (1987, 1990)], which are obtained from the pore scale Navier–Stokes equations by using volume averaging techniques, and hence are thought to be universal transport equations governing the two-phase flow through porous medium problem, can be written in the following form:

$$v_1 = -\lambda_{11}\frac{\partial \psi_1}{\partial x} - \lambda_{12}\frac{\partial \psi_2}{\partial x} \qquad (3.1)$$

and

$$v_2 = -\lambda_{21}\frac{\partial \psi_1}{\partial x} - \lambda_{22}\frac{\partial \psi_2}{\partial x}, \qquad (3.2)$$

where $\lambda_{ij} = k\,k_{rij}/\mu_i$; $i, j = 1, 2$.

Based on the experimental results presented by Bentsen and Manai (1991, 1993), it can be inferred that (Bentsen, 1998b):

$$\lambda_{ij} = \alpha_{ij}\lambda_i^0 ;\ i, j = 1, 2 \qquad (3.3)$$

where the a_{ij} are generalized partition coefficients for phase i; i, j =1, 2, and where λ_i^0, i=1, 2, are mobilities determined in a SSCO flow experiment. Upon introducing Equation (3.3) into Equations (3.1) and (3.2), one obtains:

$$v_1 = -\lambda_1^0\left(\alpha_{11}\frac{\partial\psi_1}{\partial x} + \alpha_{12}\frac{\partial\psi_2}{\partial x}\right) \tag{3.4}$$

and

$$v_2 = -\lambda_2^0\left(\alpha_{21}\frac{\partial\psi_1}{\partial x} + \alpha_{22}\frac{\partial\psi_2}{\partial x}\right). \tag{3.5}$$

Equations (3.4) and (3.5) are The Four Coefficients Transport Equations expressed in the forms of partition coefficients and the mobilities measured in a SSCO experiment.

Bentsen (1992, 1994, 1997, 1998a) has established, for all types of one-dimensional flow, that:

$$\frac{\partial P_c}{\partial x} = R_{12}\frac{\partial P_2}{\partial x} - \frac{\partial P_1}{\partial x}, \tag{3.6}$$

where R_{12} is a weak function of normalized saturation that is introduced to account for the fact that, for horizontal steady-state cocurrent flow, the pressure profile for the wetting phase is not parallel to that for the non-wetting phase [Bentsen and Manai (1991, 1993)]. Its defining equation is $R_{12} = 1 - a^*(1-S)$, where the parameter S is normalized wetting phase saturation, and the coefficient a^* must be determined experimentally.

Introducing the defining equation [Equation (3.7)] for potential

$$\psi = \rho g h + P \tag{3.7}$$

into Equation (3.6) yields:

$$R_{12}\frac{\partial\psi_2}{\partial x}-\frac{\partial\psi_1}{\partial x}=\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta, \qquad (3.8)$$

where, θ is the acute angle formed by the flowing direction of the fluid in the porous medium and the horizontal level.

Substituting Equation (3.8) into Equations (3.4) and (3.5) leads to the Modified Transport Equations for one dimensional cocurrent two phase flow:

$$v_1=-\lambda_1^0\left\{\left(\alpha_{11}+\frac{\alpha_{12}}{R_{12}}\right)\frac{\partial\psi_1}{\partial x}+\frac{\alpha_{12}}{R_{12}}\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\sin\theta\right]\right\} \qquad (3.9)$$

and

$$v_2=-\lambda_2^0\left\{(\alpha_{22}+R_{12}\alpha_{21})\frac{\partial\psi_2}{\partial x}-\alpha_{21}\left[\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2R_{12})g\text{sin}\theta\right]\right\}. \qquad (3.10)$$

As mentioned in Chapter 2, for a given saturation and potential gradient, the amount of flux flowing in a countercurrent flow experiment is less than that flowing in the equivalent cocurrent flow experiment [Lelièvre (1966), Bourbiaux and Kalaydjian (1990), Bentsen and Manai (1991, 1993)]. Such a reduction in flux in a countercurrent flow experiment, as compared to a cocurrent flow experiment can be attributed to the interfacial coupling that takes place between the two phases flowing through the porous medium. The impact of interfacial coupling on countercurrent flow can be accounted for by the interfacial coupling parameters, α_i, i =1,2. Therefore, one gets:

$$\lambda_i^*=\alpha_i\lambda_i^0;\ i=1,2, \qquad (3.11)$$

where the λ_i^*, i =1,2, are the mobilities measured in a steady state countercurrent (SSCT) flow experiment. By introducing Equation (3.11) into Equations (3.9) and (3.10), one can obtain the Modified Transport Equations for the one dimensional, countercurrent, two phase flow as follows:

$$v_1^* = -\alpha_1 \lambda_1^0 \left\{ \left(\alpha_{11} + \frac{\alpha_{12}}{R_{12}} \right) \frac{\partial \psi_1}{\partial x} + \frac{\alpha_{12}}{R_{12}} \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \right] \right\} \tag{3.12}$$

and

$$v_2^* = -\alpha_2 \lambda_2^0 \left\{ (\alpha_{22} + R_{12}\alpha_{21}) \frac{\partial \psi_2}{\partial x} - \alpha_{21} \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) \mathrm{gsin}\theta \right] \right\}. \tag{3.13}$$

3.2 Determination of the partition coefficients

For one dimensional steady state cocurrent flow, the capillary force and the gravitational force become zero, and hence Equations (3.9) and (3.10) reduce to the following forms:

$$v_1 = -\lambda_1^0 \left(\alpha_{11} + \frac{\alpha_{12}}{R_{12}} \right) \frac{\partial P_1}{\partial x} \tag{3.14}$$

and

$$v_2 = -\lambda_2^0 (\alpha_{22} + R_{12}\alpha_{21}) \frac{\partial P_2}{\partial x}. \tag{3.15}$$

Comparing Equations (3.14) and (3.15) with the traditional Muskat's transport equations which have the following forms:

$$v_1 = -\lambda_1^0 \frac{\partial P_1}{\partial x} \tag{3.16}$$

and

$$v_2 = -\lambda_2^0 \frac{\partial P_2}{\partial x}, \tag{3.17}$$

one finds that:

$$\alpha_{11} + \frac{\alpha_{12}}{R_{12}} = 1 \tag{3.18}$$

and

$$\alpha_{22} + R_{12}\alpha_{21} = 1 . \tag{3.19}$$

For horizontal steady state countercurrent flow, Equations (3.4) and (3.5) reduce to the forms shown below:

$$v_1^* = -\lambda_1^0\left(\alpha_{11}\frac{\partial P_1}{\partial x} + \alpha_{12}\frac{\partial P_2}{\partial x} \right) \tag{3.20}$$

and

$$v_2^* = -\lambda_2^0\left(\alpha_{21}\frac{\partial P_1}{\partial x} + \alpha_{22}\frac{\partial P_2}{\partial x} \right). \tag{3.21}$$

From the experiments of Bentsen and Manai (1991), for a horizontal, steady state countercurrent two phase flow, the pressure gradients of wetting and non-wetting phases have the following relationship:

$$\frac{\partial P_1}{\partial x} = -R_{12}\frac{\partial P_2}{\partial x} . \tag{3.22}$$

Substituting Equation (3.22) into Equations (3.20) and (3.21) yields:

$$v_1^* = -\lambda_1^0\left(\alpha_{11} - \frac{\alpha_{12}}{R_{12}} \right)\frac{\partial P_1}{\partial x} \tag{3.23}$$

and

$$v_2^* = -\lambda_2^0\left(\alpha_{22} - R_{12}\alpha_{21}\right)\frac{\partial P_2}{\partial x} . \tag{3.24}$$

Now, comparing Equations (3.23) and (3.24) with the traditional Muskat's transport equations for one dimensional, steady state, countercurrent, two phase flow, which are written in terms of the mobilities measured from one dimensional, steady state, cocurrent two phase flow experiment and interfacial coupling parameters in the following forms:

$$v_1^* = -\alpha_1 \lambda_1^0 \frac{\partial P_1}{\partial x} \tag{3.25}$$

and

$$v_2^* = -\alpha_2 \lambda_2^0 \frac{\partial P_1}{\partial x}, \tag{3.26}$$

one knows that:

$$\alpha_{11} - \frac{\alpha_{12}}{R_{12}} = \alpha_1 \tag{3.27}$$

and

$$\alpha_{22} - R_{12}\alpha_{21} = \alpha_2. \tag{3.28}$$

Combining Equations (3.18), (3.19), (3.27) and (3.28) and solving them for the partition coefficients, one gets:

$$\alpha_{11} = \frac{1+\alpha_1}{2}, \tag{3.29}$$

$$\frac{\alpha_{12}}{R_{12}} = \frac{1-\alpha_1}{2}, \tag{3.30}$$

$$\alpha_{22} = \frac{1+\alpha_2}{2}, \tag{3.31}$$

and

$$R_{12}\alpha_{21} = \frac{1-\alpha_2}{2}. \qquad (3.32)$$

By introducing Equations (3.29), (3.30), (3.31) and (3.32) into Equations (3.9), (3.10), (3.12) and (3.13), one obtains the Modified Transport Equations for cocurrent two phase flow in the form of Equations (3.33) and (3.34):

$$v_1 = -\lambda_1^0 \left\{ \frac{\partial \psi_1}{\partial x} + \left(\frac{1-\alpha_1}{2} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \right] \right\} \qquad (3.33)$$

and

$$v_2 = -\lambda_2^0 \left\{ \frac{\partial \psi_2}{\partial x} - \left(\frac{1-\alpha_2}{2R_{12}} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g\mathrm{sin}\theta \right] \right\}, \qquad (3.34)$$

and the Modified Transport Equations for countercurrent two phase flow in the form of Equations (3.35) and (3.36):

$$v_1^* = -\alpha_1 \lambda_1^0 \left\{ \frac{\partial \psi_1}{\partial x} + \left(\frac{1-\alpha_1}{2} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \right] \right\} \qquad (3.35)$$

and

$$v_2^* = -\alpha_2 \lambda_2^0 \left\{ \frac{\partial \psi_2}{\partial x} - \left(\frac{1-\alpha_2}{2R_{12}} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g\mathrm{sin}\theta \right] \right\}. \qquad (3.36)$$

3.3 Interfacial coupling

As mentioned in Section 2.2.3.3, interfacial coupling has two major components, viscous coupling and capillary coupling. The viscous coupling arises due to the viscous drag exerted by one fluid on the other when they flow through the same porous medium and hence should be associated with the mobility of the fluids; the capillary coupling arises due to coupling, through the capillary function, of pressure across the interfaces of the fluids and must be associated with the potential gradients acting across the flowing phases [Bentsen (2005)].

The mobility and the potential gradient of the fluids appears as a product in the transport equations; consequently, the viscous coupling and capillary coupling parameters should show up as a product. Therefore, the interfacial coupling can be defined as follows:

$$\alpha_i = \alpha_{vi}\alpha_{ci} \qquad i\text{=}1, 2, \tag{3.37}$$

where, the α_{vi} are the viscous coupling parameters, and the α_{ci} are the capillary coupling parameters. The defining equations for viscous coupling and capillary coupling parameters are presented below.

3.3.1 Viscous coupling

3.3.1.1 Defining equation of viscous coupling

Based on Onsager's reciprocity relations [Onsager (1931a and 1931b)], it can be assumed that the cross terms in Equations (3.1) and (3.2) are equal to each other [Rose (1988), Kalaydjian (1990), Liang and Lohrenz (1994)]:

$$\lambda_{12} = \lambda_{21}\,. \tag{3.38}$$

To satisfy Equation (3.38), in view of Equations (3.3), (3.30) and (3.32), Bentsen (2003) constructed the defining equations for viscous coupling as follows:

$$\alpha_{v1} = 1 - \frac{c}{R_{12}}\frac{\lambda_2^0}{\lambda_m^0} \tag{3.39}$$

and

$$\alpha_{v2} = 1 - cR_{12}\frac{\lambda_1^0}{\lambda_m^0}\,, \tag{3.40}$$

where the parameter, λ_m^0, is introduced to ensure dimensional consistency and also has to satisfy the following requirements. When the normalized wetting phase saturation, *S*, equals one, $\lambda_m^0 = \lambda_{1r}^0$; when *S* equals zero, $\lambda_m^0 = \lambda_{2r}^0$, where, λ_{1r}^0 and λ_{2r}^0 are the endpoints mobilities for the wetting and non-wetting phase, respectively. A defining equation which satisfies these requirements has the following form:

$$\lambda_m^0 = S\lambda_{1r}^0 + (1-S)\lambda_{2r}^0 . \qquad (3.41)$$

The parameter *c* controls the amount of viscous coupling and it has to be determined experimentally.

3.3.1.2 Determination of the parameter *c*

As pointed out by Bentsen (1998b), the effect of viscous coupling is to increase the curvature of the normalized, countercurrent relative permeability curves, as compared to the normalized, cocurrent relative permeability curves. Therefore, theoretically, it is possible to determine the parameter *c* by conducting two sets, one cocurrent and one countercurrent, of experiments. However, it is very hard, if not impossible, to achieve this experimentally for the following two reasons. Firstly, the magnitude of the viscous coupling effect is so small [Zarcone and Lenormand (1994), Rakotomalala et al. (1995)] that the differences in curvature arising out of using two experiments would be large enough to mask the small changes in curvature that are caused by viscous coupling. Secondly, the curvature of the normalized relative permeability curves can differ significantly in replicated experiments [Bentsen and Manai (1991, 1993)].

Nevertheless, based on the results presented by Zarcone and Lenormand (1994) and Rakotomalala et al. (1995), Ayodele (2004) found out that the maximum value of parameter *c* is about $2\phi^2$ provided that the porosity, ϕ, is less than 0.35. Thereafter, he found that the value of *c* was $0.001\phi^2$ when he used curve fitting techniques in conjunction with the cocurrent and countercurrent experimental results [Data Group A and Data Group C in the thesis by Ayodele (2004)]. Due to the reasons stated in the previous paragraph, it is hard to say how much credibility should be given to the second value of *c*, $0.001\phi^2$, given by Ayodele (2004). Therefore, it was decided to accept $2\phi^2$ as the value of parameter *c* in the simulation analysis presented in this study.

3.3.2 Capillary coupling

By replacing the multiphase porous medium by three overlapping continua, and by using the concept of a representative elementary volume (REV) [Bear (1972)], Bentsen (2005) constructed the following defining equations for the capillary coupling parameters:

$$\alpha_{c1} = \alpha_{c2} = \alpha_c = 1 - \phi \,. \tag{3.42}$$

3.4 Validation methods

In view of Equations (3.33), (3.34), (3.35) and (3.36), it can be determined that these equations can be validated completely by conducting three pairs of experiments, which are given below:

(a) Horizontal, steady state, cocurrent (SSCO) flow experiments and horizontal, steady state countercurrent (SSCT) flow experiments,

(b) Horizontal, steady state cocurrent (SSCO) flow experiments and unsteady state cocurrent (USCO) flow experiments conducted at various flowing angles, *θ*,

(c) Horizontal, steady state cocurrent (SSCO) flow experiments and unsteady state countercurrent (SSCT) flow experiments conducted at various flowing angles, *θ*.

Moreover, each pair of experiments has to be conducted using fluids with different viscosity ratios, μ_1 / μ_2, and porous media with different porosities.

In the first pair of experiments, Equations (3.33), (3.34), (3.35) and (3.36) degenerate to Equations (3.16), (3.17), (3.25) and (3.26), respectively. One can easily compare the mobilities measured form SSCO and SSCT experiments and see if the coupling parameters are well defined or not.

In the last two pairs of experiments, one can obtain the mobilities, λ_1^0 and λ_2^0, from a horizontal, SSCO flow experiment and then insert them, together with the coupling parameters, into Equations (3.33), (3.34), (3.35) and (3.36) under the conditions of both unsteady state and various flowing angles, *θ,* so that these equations can be solved. By comparing the solutions of these equations and the measured experimental data such as pressure profiles and saturation profiles, it can be determined that if the second term in the big bracket on the right hand side (RHS) of these equations is organized properly or not.

Obviously, it needs a lot of work to totally validate these equations. Due to the time issue and the limitations of material and experimental equipment, this study just focuses on the second pair of experiments, and only the horizontal SSCO experiment and the horizontal USCO experiment are conducted. The influences of the flowing angle and viscosity ratio are investigated by utilizing the simulator developed on the basis of the Modified Transport Equations.

CHAPTER 4

Experimental description and discussion

4.1 Introduction

To investigate the interfacial coupling effect in USCO flow situations, several experiments were conducted. Two sets of experimental data from these experiments are presented and discussed in this chapter. The first set of experimental data was acquired using the equipment developed by Ayub and Bentsen (2000) and slightly modified by Ayodele (2004). Later, with the goal of improving the accuracy and efficiency of the experimental measurements, the data acquisition system and some related hardware were upgraded, and then a second set of experimental data was obtained.

The description of the experimental equipment, the modification of the equipment, the experimental procedures and the discussion of the experimental results are presented in the flowing sections.

4.2 Experimental equipment and modifications

As shown in Figure 4-1, the experimental equipment consists of a core holder with end caps, a fluid injection system, an effluent collection system, a dynamic pressure measurement system, a dynamic saturation system and a data acquisition system. A detailed description of the equipment can be found in the paper by Ayub and Bentsen (2000). For the reader's convenience, and for ease of discussion of some problems related to the experiment and corresponding modifications, these components are described here, briefly, or in detail.

4.2.1 Core holder and end caps

There are several core holders, of various lengths, available in the laboratory. In order to (a) measure more pressure points and, in turn, to reduce the uncertainty in

Equipment & Accessories

1. Core-holder
2. Pressure ports
3. Water saturation sensor
4. Pulse dampener
5. Oil injection pump
6. Water injection pump
7. Water reservoir
8. Oil reservoir
9. Stepping motor
10. Motor controller
11. Chain driven track
12. Oil collector
13. Water collector
14. Oil balance (out)
15. Water balance (out)
16. Transducer cable junction
17. Frequency counter
18. DC Power supply
19. AC Line conditioner
20. Control & data acquisition
21. 110 VAC Power supply
22. Co-axial cable
23. DC Power to sensor
24. Data acquisition cables
25. Oil pump controller
26. Water pump controller
27. AC Power cables
28. Oil balance (inj)
29. Water balance (inj)
30. Byteway, the balance cables junction

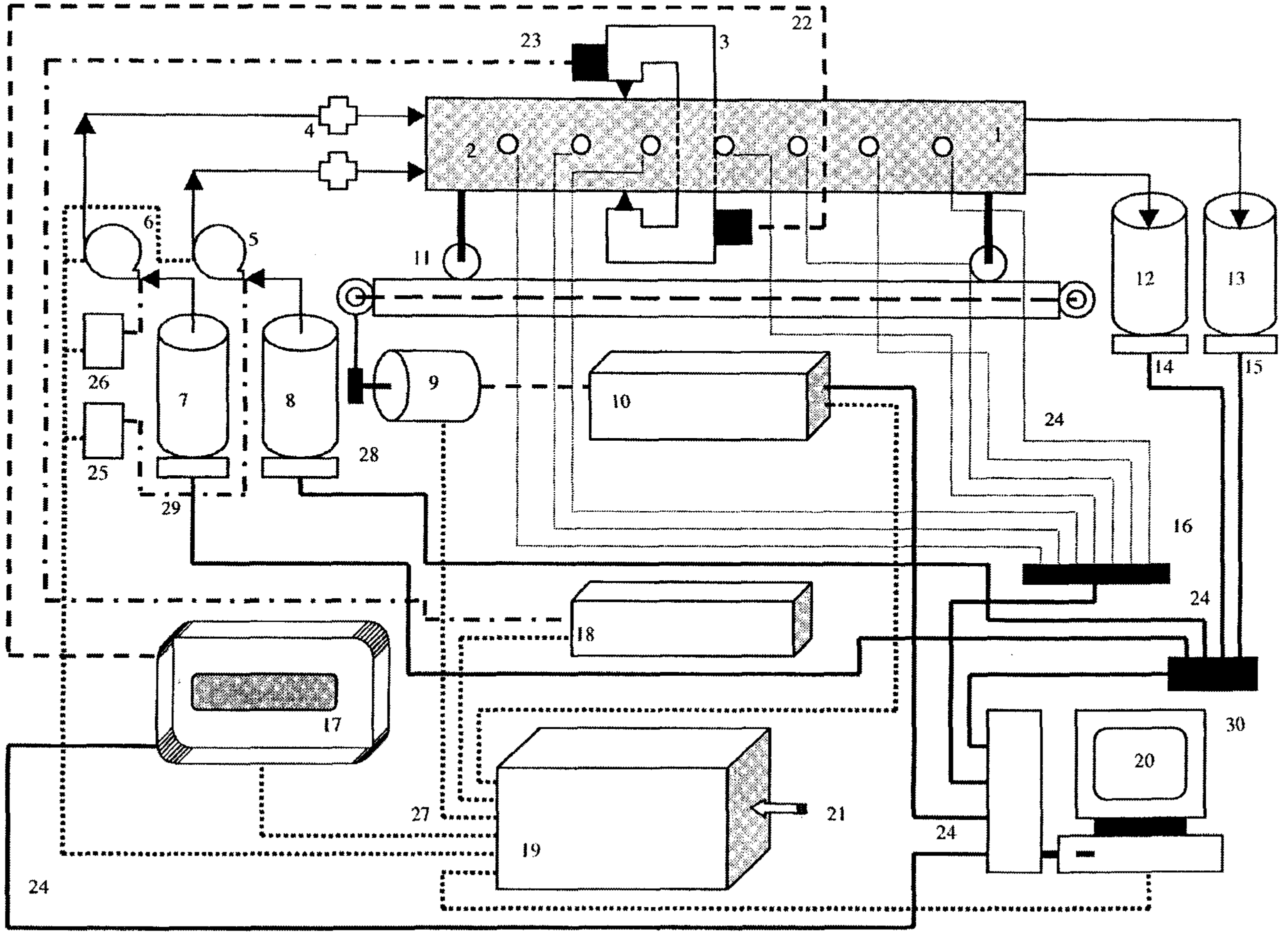


Figure 4-1: Schematic representation of experimental equipment [Courtesy: Ayub and Bentsen (2000)]

analyzing the pressure profiles, and (b) eliminate end effects during the experiment as much as possible, the longest one was used in the experiments carried out for the current study. The schematic of this core holder is shown in Figure 4-2. There are 14 pressure ports, 7 on each side of the core holder. Seven of them were used to measure wetting phase pressures, and the remaining seven were used for non-wetting phase pressure measurement.

Due to limitations in the design of the cart that holds the core holder, the dynamic saturation measurement system could not reach the inlet and outlet face of the core. The maximum saturation measurement interval was 56 cm, from 10 cm to 66 cm away from the inlet face of the core. The benefit of this limitation is that the 10 cm and 17.6 cm lengths of the core at the inlet and outlet ends of the core, respectively, as shown in Figure 4-2, can be used to reduce the end effects.

Three types of end caps were used in the experiments. They are referred to as Type A, Type B and Type C in this study, and they are shown in Figures 4-3 and 4-4, respectively. Type A end caps were installed on both ends of the core holder when conducting SSCO flow experiments. A combination of Type A and Type B [with fritted Disc (porous glass disc) attached] end caps, or a combination of Type A and Type C (with Teflon Disc attached) end caps were used when doing USCO flow experiments, in which water was displacing oil, or when preparing a core for USCO experiments, in which oil was displacing water. A Type A end cap was always installed at the outlet end of the core, while a Type B or Type C end cap, depending on experimental requirements, was used at the inlet end of the core.

The fritted and Teflon discs used in the end caps were supposed to be water wet and oil wet material, respectively. At the inlet end of the core, these discs were used to distribute evenly the fluids, water and/or oil, before the fluids come in contact with the core by utilizing the capillary force to overcome the gravity force so as to eliminate the inlet end effect. At the outlet end, a Type A end cap was always used to provide a passage way for fluids (water and/or oil) to flow out of the core and, at the

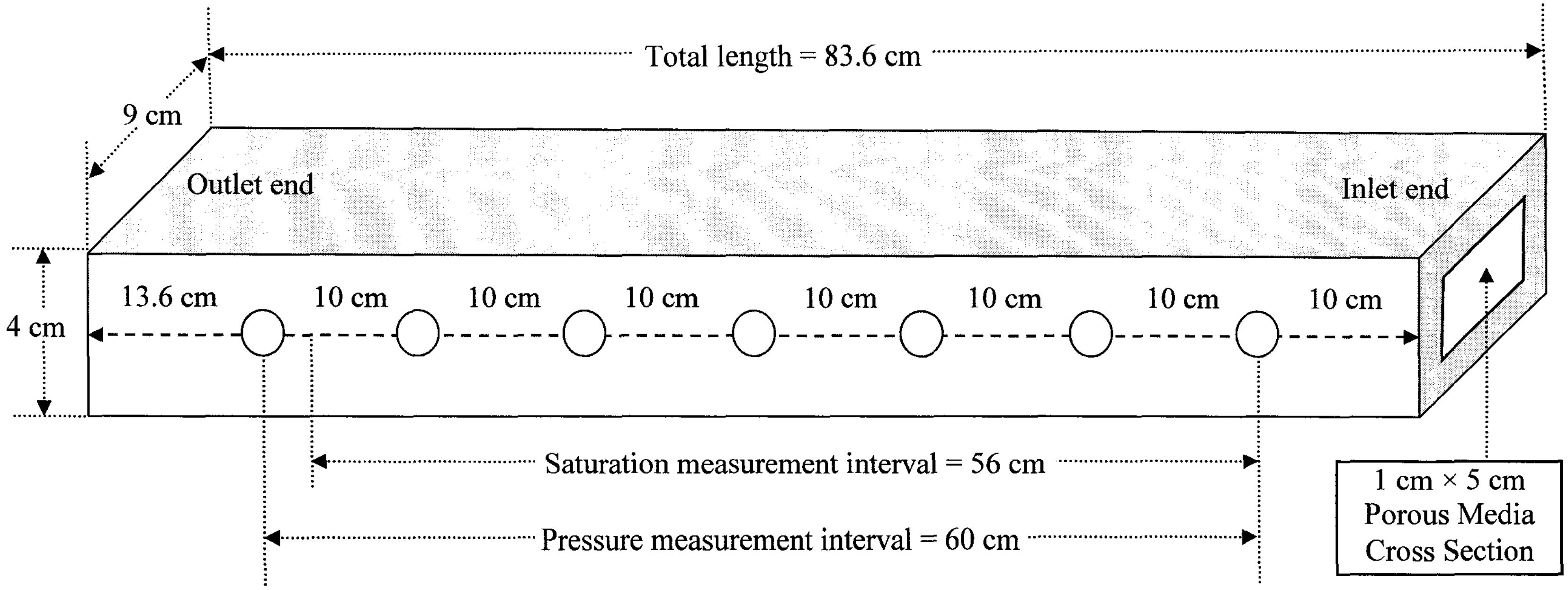


Figure 4-2: Configuration of the core holder

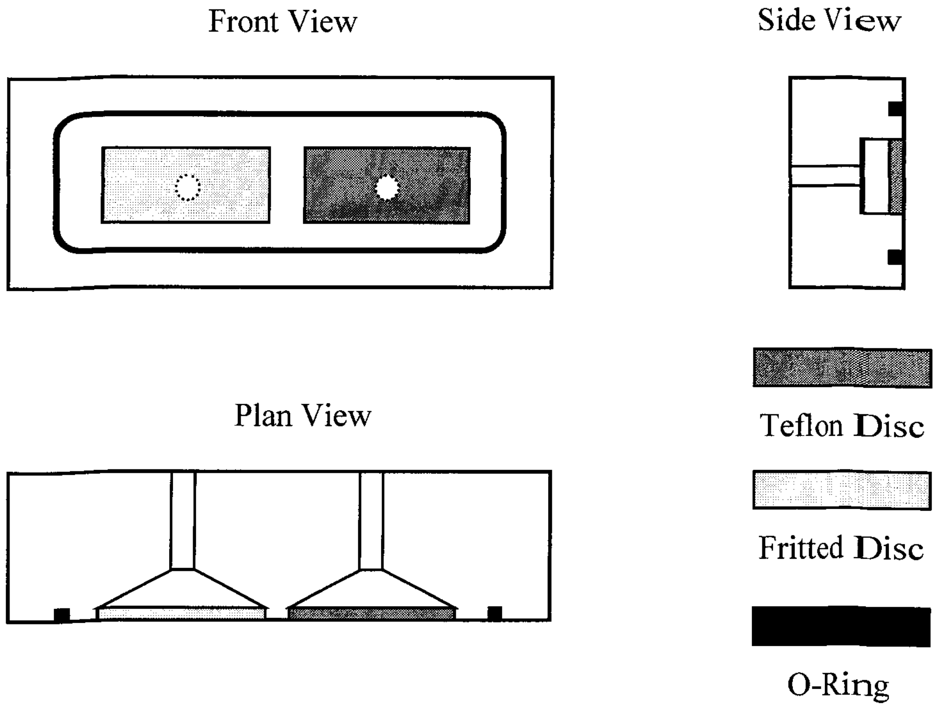


Figure 4-3: Schematic of Type A end cap

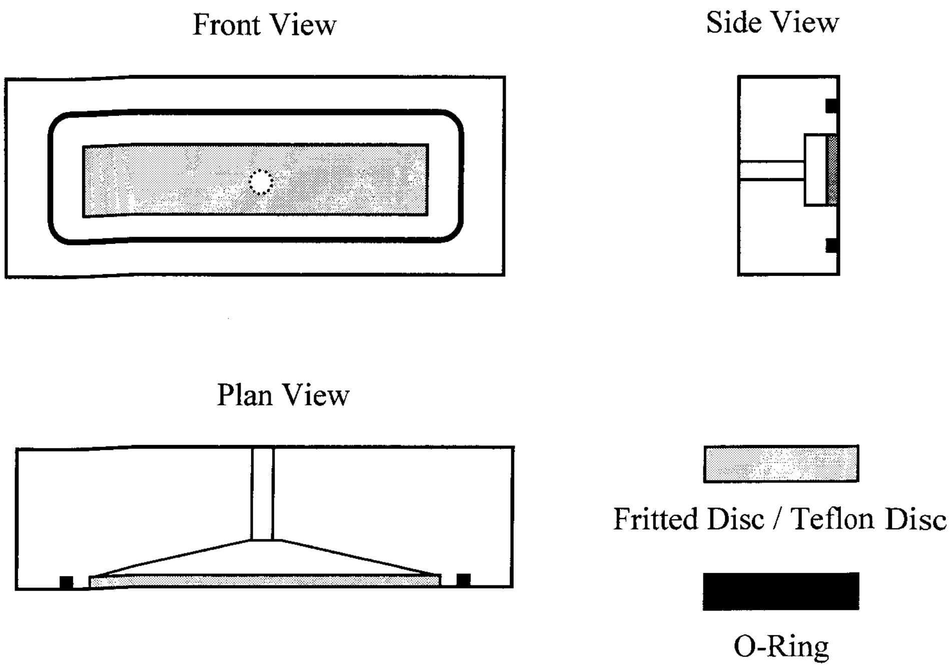


Figure 4-4: Schematic of Type B (Fritted Disc) and Type C (Teflon Disc) end caps

same time, to prevent the unconsolidated sand from being flushed out of the core holder.

4.2.2 Fluid injection and effluent collection system

The fluid injection system consisted of two FMI (Fluid Metering Inc.) "Q" Model pumps, two FMI pulse dampeners and two electronic balances (manufactured by Setra System Inc.). The FMI "Q" Model pump was able to provide a variety of flow rates by adjusting the stroke length and stroke rate. The FMI pulse dampener was installed after the injection pump and before the inlet end cap to achieve pulse free fluid flow. The electronic balance was used to weigh the injection fluid with respect to time.

The effluent collection system consisted of two graduated cylinders and two electronic balances (manufactured by Mettler Toledo Company). It was found during the experiments that the fritted disc installed on the outlet end cap (Type A end cap) could not prevent oil from flowing through, and the Teflon disc could not prevent water from flowing through. It was decided to put all the effluent fluids (water and/or oil) into the same graduated cylinder sitting on the electronic balance. In this way, one can use the weight of the mixed fluids and the volume of one fluid to determine the weight or volume of the other fluid. The weight or volume of the fluids was needed when the material balance calculation was performed to determine the water saturation of the core, as discussed in Section 4.2.4.2.

4.2.3 Dynamic pressure measurement system

Instead of measuring static capillary pressure and the mixture pressure (or non-wetting phase pressure) in two phase flow to obtain the wetting phase pressure, in this study, an attempt was made to measure the phase pressure of the two flowing phases directly, and then determine the capillary pressure, which, it needs to be noted, is a dynamic capillary pressure.

To measure the phase pressure, 14 Validyne Model DP15 variable reluctance pressure transducers, together with Dash 44 diaphragms (Validyne Engineering Corp.), were used. Seven of them, with Teflon discs (manufactured by Knotes Scientific Glassware/Instrument) mounted, referred to as oil phase pressure transducers, were installed on one side of the core holder to measure the oil phase pressure. The other 7 pressure transducers, mounted with Teflon discs (manufactured by Knotes Scientific Glassware/Instrument) with a lot of punched holes and MF-Millipore hydrophilic membranes (MF disc Mixed Cellulose Esters, provided by Millipore Corporation), referred to as water phase pressure transducers, were installed on the opposite side of the core holder to measure the water phase pressure.

4.2.3.1 Principle of dynamic pressure measurement

Before discussing dynamic pressure measurement, it is important to review the principles underlying the use of variable reluctance pressure transducers to measure pressure. A typical variable reluctance pressure transducer, such as a Validyne Model DP15, consists of a diaphragm of magnetically permeable stainless steel clamped between two blocks of stainless steel. Embedded in each block is an inductance coil on an E-shaped core. In the undeflected position, the diaphragm is centered with equal gaps between it and the legs of each E-core to provide equal reluctances for the magnetic flux of each coil. A pressure difference applied through the pressure ports deflects the diaphragm toward the cavity with the low pressure, decreasing one gap and increasing the other. As the magnetic reluctance varies with the gap and determines the inductance value of each coil, the diaphragm deflection increases the inductance of one coil and decreases that of the other (provided by Validyne Engineering Corp.). Consequently, the pressure difference can be measured.

Moreover, the dynamic pressure response can be measured by a variable reluctance pressure transducer provided that (a) there is no pressure drop through the passage from the pressure source to the cavity formed by the diaphragm and the transducer body, or this pressure drop is negligible, and (b) the flow rate of the material conveying pressure is high enough to deflect the diaphragm and get equilibrium immediately. Of course, condition (b) implies that there is no air trapped in the cavity.

4.2.3.2 Dynamic water phase pressure measurement

Dynamic water phase pressure measurement has been a problem for a long time [as reported by Ayub (2000) and Ayodele (2004)]. One objective of this study was to find out what causes the problem and to resolve it.

In order to measure dynamically the water phase pressure in a porous medium that contains both water and oil, based on the discussion in Section 4.2.3.1, one has to create a way to let water, but not oil, enter the pressure transducer to deflect the diaphragm. Moreover, conditions mentioned in Section 4.2.3.1 must be satisfied.

Theoretically, this can be achieved by covering the fluid intake port of the pressure transducer with a highly water wet material that has a high permeability to water and a high threshold pressure to oil. The high threshold pressure to oil is able to prevent oil from entering the pressure transducer, while the high permeability to water allows water to flow through this material without causing too much pressure drop. It needs to be noted that a prerequisite of this solution is that the water flow rate in the porous medium has to be high; otherwise, no matter how perfect the water wet material is, condition (b) set out in Section 4.2.3.1 can not be met. It is well known that the water phase flow rate in the porous medium may be affected by many factors such as water saturation, properties of the core and so on.

Practically, however, it is difficult to find such a material. Ayub (2000) tried a lot of things such as fritted discs manufactured locally, fritted discs manufactured by Corning Corp., and several types of hydrophilic filter membranes used by Hammervold et al. (1998), Hammervold and Skjaeveland (1992), and Longeron et al. (1994). Finally, he decided to use the fritted discs manufactured by the Corning Corp. as a support base, above which a hydrophilic filter membrane manufactured by Millipore Corporation was attached. Nevertheless, this was still not a satisfactory solution, as mentioned in the theses of Ayub (2000) and Ayodele (2004).

In the present study, to seek a better solution for dynamic water phase pressure measurement, several things were tried. Firstly, the Super Hydrophilic Membranes manufactured by Filtration Solutions Inc. were tested. These membranes really have a higher threshold pressure to oil as compared to the membranes made by the Millipore Corporation. However, these membranes were not selected due to their low permeability.

Secondly, as suggested by Ayub (2000) and the ARC (Alberta Research Council), the fritted disc, used as the material to cover the fluid intake port of the pressure transducer, was treated by using a NaOH solution to improve the hydrophilic ability (improve the threshold pressure to oil). This approach did not make any visible difference. Moreover, it was found during experiments that the fritted discs manufactured by Corning Corp. were too dense for water to flow through effectively. This led to the result that the water phase pressure transducers could not even sense the pressure change under low pressure conditions.

Finally, it was discovered that a possible solution may be to use the MF-Millipore hydrophilic membranes (MF disc Mixed Cellulose Esters, provided by Millipore Corporation) as the filter material, under which a high porosity and high permeability disc (Teflon disc with a lot of punched holes in it) was installed in the fluid intake port of the pressure transducer to provide a support base. To prevent the hydrophilic membranes from changing wettability, as noted in the thesis by Ayub (2000), the water phase pressure transducers were installed just prior to when pressure measurements were needed, and uninstalled while injecting fluid to prepare the core for the experiment. In addition, in order to avoid air being trapped in the cavity of the transducer, this cavity was filled fully with water before the installation of the transducer onto the core holder. This method was better than bleeding off the air in the cavity after installation, because in the bleeding process, one can not avoid oil penetrating into the membrane and the support base and creating a flow path into the cavity of the transducer. Nevertheless, as is seen in Section 4.4.2.2.2, the pressure profiles measured during an experiment were still not as good as expected, which also can be attributed to the influence of the properties of the porous medium system discussed below, as well as the unsatisfactory behavior of the selected hydrophilic membranes.

Generally speaking, the porous medium system has two main effects on the dynamic water phase pressure measurements. Firstly, if the flow rate of the water phase fluid in the porous medium is too low, it will take a while for the diaphragm to be deflected, no matter how perfect the hydrophilic membranes attached to the pressure transducers are. Under this operating condition, there must be some difference between the real water phase pressure in the porous medium and the measured pressure at a certain point in time. The difference is caused by the time period needed to achieve equilibrium. However, it is hard to tell how large this influence is. In the experiments, this effect was assumed to be negligible.

Secondly, under the irreducible water saturation condition, if one assumes that the core is isotropic and homogenous, as is expected, the hydrophilic membranes, which are attached to the water phase pressure transducers and in contact with the porous medium, must be covered mostly by oil instead of by water. Because the core is a water wet system, as it should be, the oil phase, which has a higher pressure than the water phase, will try to penetrate the hydrophilic membranes and flow into the transducer's cavity, especially when the water, which is immobile, can not compete with it. If the wetting properties of the hydrophilic membranes, such as contact angle and porosity, are different from those of the porous medium (in most cases, they are different), no matter whether or not the oil can enter the water phase pressure transducers, the measured water phase pressure will not be the real water phase pressure in the core any more. The calculated capillary pressure obtained by using measured pressures is probably the capillary pressure of the hydrophilic membranes instead of that of the core.

To overcome this problem as much as possible, it was decided to fill a small amount of water (about 1 cc) into the pressure port of the core holder before installing the water phase pressure transducer so as to create a water barrier between the porous medium and the hydrophilic membranes. This method can reduce also the chance for air to be trapped in the core during the installation of the pressure transducer.

From what has been discussed above, it is obvious that there are many factors influencing the measurement of pressure; consequently, it is extremely hard to measure the dynamic water phase pressure accurately and, moreover, the magnitude of the measurement error is not easily determined. However, from an engineering perspective, the measured water phase pressure profiles, which can be found in Section 4.4.2.2.2, were adequate, even though they were not as good as expected.

4.2.3.3 Dynamic oil phase pressure measurement

As can be seen from both the literature [Ayub (2000) and Ayodele (2004)] and experiments conducted for the current study, the dynamic oil phase pressure measurement is much more easily undertaken than the dynamic water phase pressure measurement. Such is the case because the oil wet filter material, which is mounted in the fluid intake port of the pressure transducer to measure the oil phase pressure, just needs to have a high permeability to oil instead of simultaneously having both a high permeability to oil and a high threshold pressure to water. This is because, in a water wet porous medium system, the oil phase pressure is always higher than the water phase pressure, and, hence, has the preferential ability to enter the oil pressure transducers. Therefore, one does not need to seek a balance between high permeability and high threshold pressure, which is hard to obtain because a high permeability usually implies high porosity, while a high threshold pressure usually implies low porosity.

Moreover, unlike the dynamic water phase pressure measurement, the properties of the porous medium system have less effect on the oil phase pressure measurement. Even at the residual oil saturation, where the water in the core covers most of the surface of the Teflon disc on the oil pressure transducer, the oil phase pressure measurement is not affected, unless the surface of the Teflon disc is completely covered by oil, which, sometimes, can be the case due to the local heterogeneity of the core. However, low oil flow rates inside the core can still cause some difference between the real oil pressure in the core and the measured pressure at a certain point in time, which is caused by the time period needed to achieve equilibrium.

It was found that Validyne Model DP15 variable reluctance pressure transducers, together with Dash 44 diaphragms (Validyne Engineering Corp.) with Teflon discs (manufactured by Knotes Scientific Glassware/Instrument) mounted at the fluid intake port of the pressure transducer, were suitable for the dynamic oil phase pressure measurements.

4.2.3.4 The arrangement of the pressure transducers on the core holder

Generally, there are two ways to arrange the pressure transducers on the core holder. The first way is to install 7 water pressure transducers on one side of the core holder and 7 oil pressure transducers on the opposite side of the same core holder. The second way is to install the water and oil pressure transducers alternatively on both sides of the core holder. Ayub (2000) determined that these two different arrangements of the pressure transducers on the core holder gave the same pressure results. Therefore, for operating convenience, the first arrangement was used in the experiments conducted for the current study.

4.2.4 Dynamic saturation measurement system

The dynamic saturation measurement system built by Ayub (2000) was used in the current experiments. The main part of this system is a resonator (as shown in Figure 4-5) whose dielectric constant changes as the water saturation in the core placed between its parallel plates changes. The method used to measure saturation is basically a capacitance (described by dielectric constant) based method.

Because the relative dielectric constant of the sand, the oil and the core holder (made of acrylic) is so small as to be negligible, when compared to that of water, the change of the relative dielectric constant of the core (sand, oil and core holder) is considered

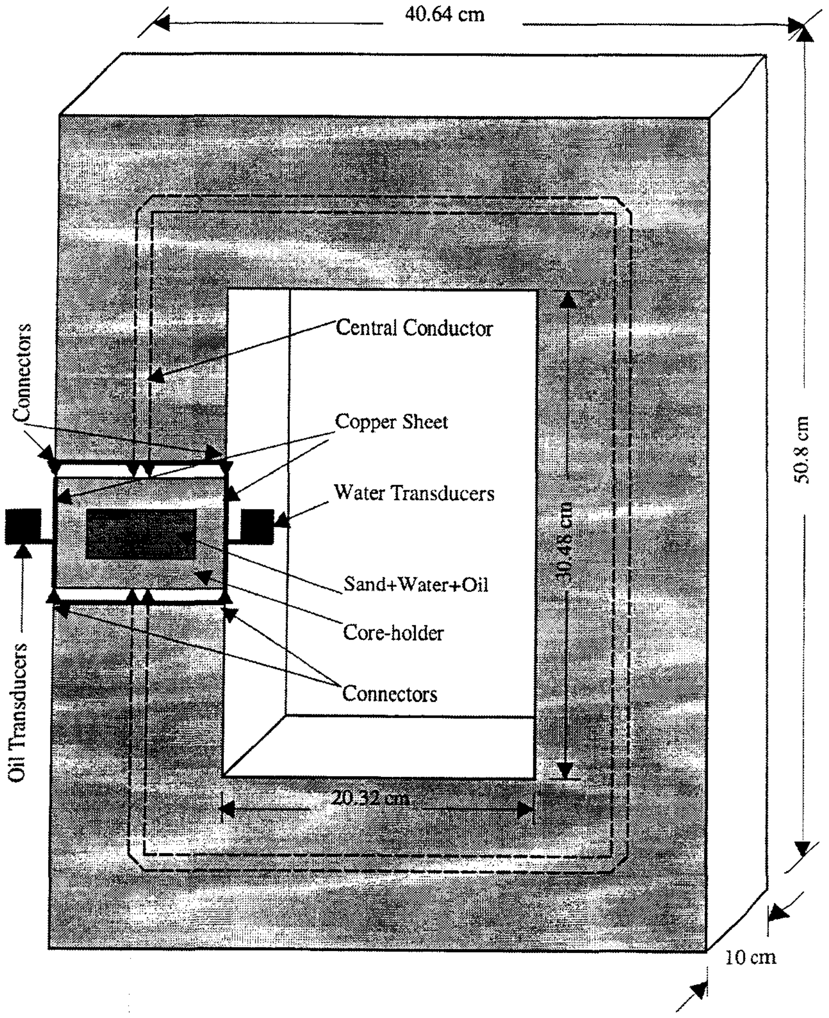


Figure 4-5: Resonator for the saturation measurement system with cross-sectional view of the core holder [Courtesy: Ayub and Bentsen (2000)]

to be the main indicator of the change in the amount of water in the core. Moreover, the change in the dielectric constant causes a change in frequency. Therefore, one can establish a relationship between the water saturation in the core and the frequency that can be measured using a frequency counter connected to the resonator through a certain circuit (see Figure 4-6).

Ayub (2000) used an empirical relationship to determine the saturation profile by simplifying the equation presented by Orlov (1970). This empirical relationship is given as:

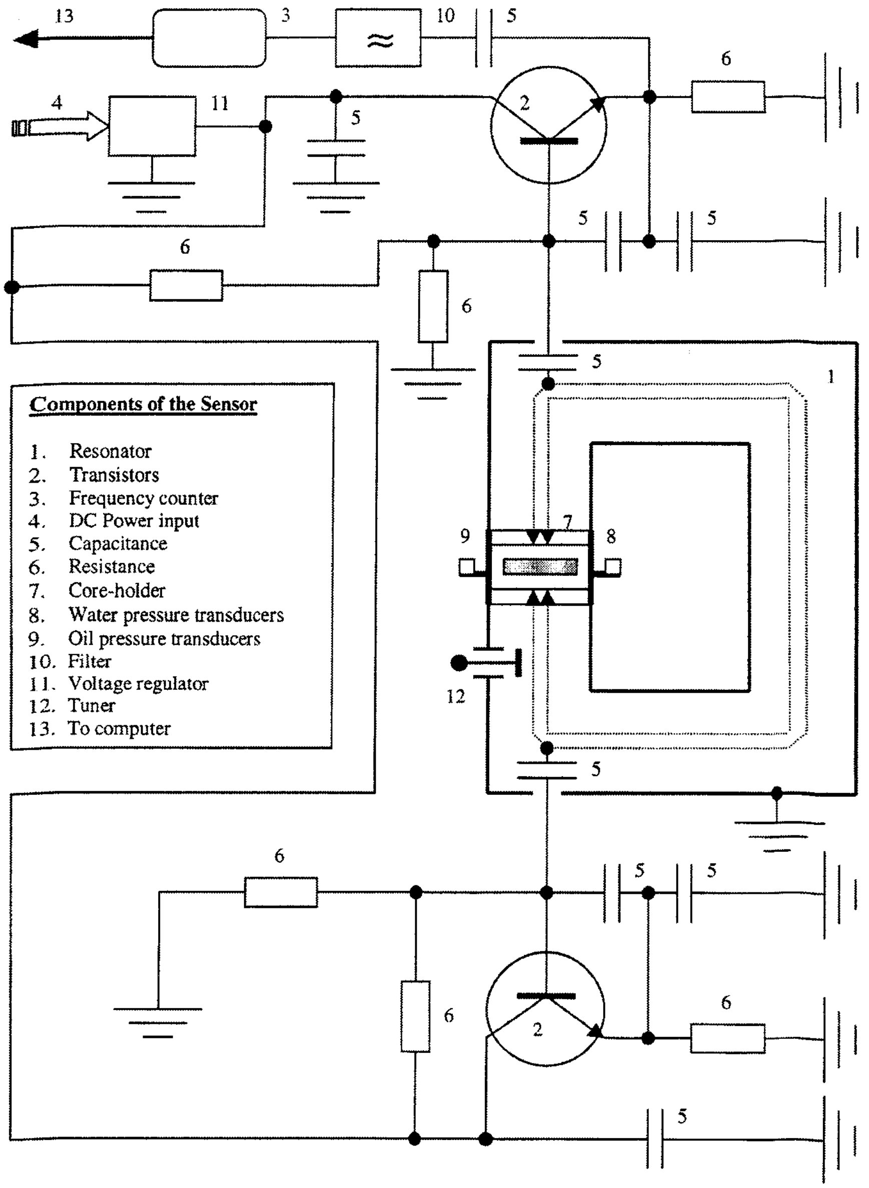


Figure 4-6: Circuit diagram of the water saturation measurement sensor [Courtesy: Ayub and Bentsen (2000)]

$$S_1 = \frac{1}{A_o f_o \tan(B_o f_o)} \, . \qquad (4.1)$$

In Equation (4.1), f_o is the frequency response, and S_1 is the water saturation in the core. The parameters A_o and B_o can be determined by non-linear regression analysis, once the frequency response against various water saturation levels has been determined experimentally.

4.2.4.1 The relationship between water saturation and frequency

The relationship between water saturation and frequency depends on many factors and several results have been reported at various frequency levels [Davis (1980), Kraszewski (1996), Berg (1995), Nguyen et al. (1999), Adisoemarta (2000), Bona et al. (2001 and 2002) and West et al. (2003)], As mentioned in the work by Ayodele (2004), no final consensus exists on the relation between water saturation and frequency.

Davis (1980) and Kraszewski (1996) observed that the anomalous response of fluids in a porous medium system in the frequency range from 100MHz to 200MHz was found to be minimal. Therefore, Ayub (2000) conducted experiments to measure the dynamic water saturation using this frequency range, and used the empirical relationship described by Equation (4.1) to fit the experimental data.

Later on, some electrical components of the resonator were changed [Ayodele (2004)]. As a result, the operating frequency range of the resonator changed to about 82 to 88 MHz. This frequency range is thought to be acceptable for determining the water saturation, which is supported by the fact that the experimental results presented by Nguyen et al. (1999), obtained from core flooding experiments, show that the frequency response to various saturation levels can be observed in the frequency range from 82 to 88 MHz.

Using this new frequency range to conduct experiments, Ayodele (2004) found that the relationship between water saturation and frequency can be described by the following inverse third order polynomial function:

$$S_1 = a + \frac{b}{f_o} + \frac{c}{f_o^2} + \frac{d}{f_o^3}, \qquad (4.2)$$

where, a, b, c and d are coefficients determined by non-linear regression analysis.

Since the modified resonator was used in the experiments for the current study, Equation (4.2) was also employed to analyze the experimental data, as can be seen in Sections 4.2.4.2 and 4.2.4.3.

4.2.4.2 Calibration of the dynamic saturation measurement system

Different cores have different properties, such as sand properties, porosity, and local heterogeneity and so on, which lead to different values for the coefficients found in Equation (4.2). Therefore, one needs to calibrate the dynamic saturation measurement system with a prepared core, before running core flooding experiments.

Usually, the calibration can be achieved by conducting SSCO flow experiments in which oil and water are injected simultaneously at different ratios of the flow rates. During SSCO flow experiments, the water saturations along the core should be a constant value. With the help of a volumetric material balance technique, one can determine easily the exact value of the water saturation corresponding to different ratios of the flow rates. Moreover, the frequency responses at different locations along the core for each water saturation level were recorded. With several sets of water saturation data, and the corresponding frequency values, a unique calibration equation in the form of Equation 4.2 can be determined.

4.2.4.3 Problems encountered and corresponding solutions

The frequency profiles shown in Figure 4-7 were acquired from SSCO flow experiments conducted at various water saturation levels. As can be seen, these frequency profiles are almost parallel to one another. However, they are not perfectly horizontal straight lines, as they are supposed to be. This can be caused by the following factors.

Firstly, geometrical problems associated with the construction of the equipment and the core holder were likely the main influence factors. As shown in Figures 4-1 and 4-5, the contact between the core holder and the water saturation sensor, which affects the frequency responses, may change if (a) the surface of the core holder is not perfectly smooth, (b) the cross-section of the core holder is not perfectly rectangular, (c) the position of either the core holder or the chain driven track is not perfectly horizontal and (d) the driving chain is so long that it can vibrate when the stepping motor is running.

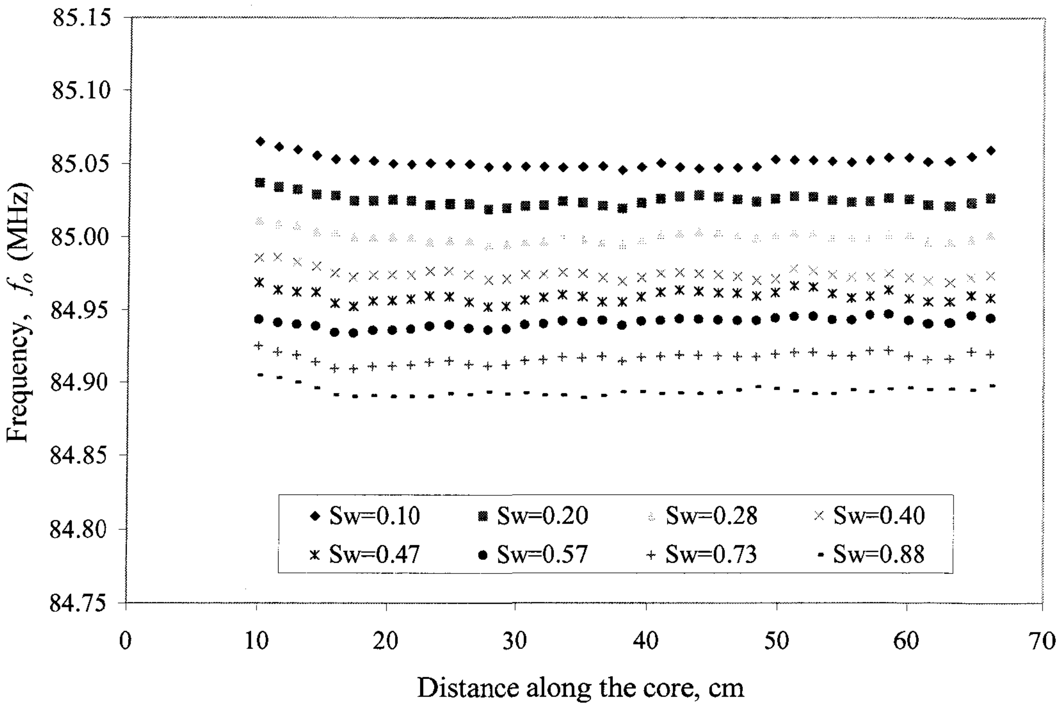


Figure 4-7: Frequency profiles measured at various water saturation levels during SSCO flow experiment

Secondly, the local core heterogeneity can be another influence factor, although care was taken to minimize local heterogeneities when the core was being packed.

Thirdly, there were also some minor influence factors such as electromagnetic interference, and temperature and humidity variations in the laboratory.

Out of these three types of influence factors, the third one was thought to be so small as to be negligible, because attention was paid to keeping the environment of the laboratory as stable as possible when an experiment was being conducted. The impact of the influence factors in the first two categories were location and/or direction related. This is also the reason why the frequency profiles corresponding to various water saturation levels were parallel to one another, even though they were not perfectly horizontal straight lines.

The location and/or direction related influence can be eliminated by calibrating the water saturation sensor at each location along the core in a certain direction. In other words, there is a unique relationship, corresponding to the core moving direction, between the frequency responses and the water saturation at each location along the core. For example, in the experiment conducted, the frequency responses were recorded at 39 measuring locations from 66 cm through 10 cm away from the inlet end of the core when the core was moving from the outlet end towards the inlet end. Hence, there were 39 calibration equations associated with 39 measuring locations. The coefficients of these equations are given in Appendix A (To facilitate the use of the experimental data measured in this study, all of the data sets collected are burned into the CD attached at the back of this thesis). The calibration curve corresponding to the location 10 cm away from the inlet end of the core is shown in Figure 4-8 for illustration purposes.

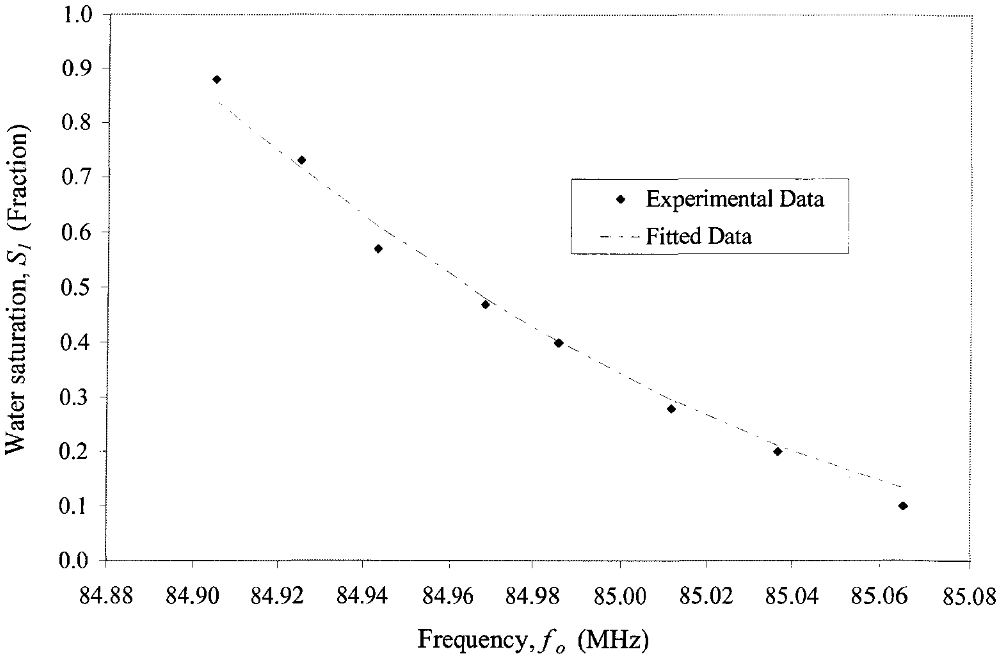


Figure 4-8: Calibration curve to convert the frequency responses into water saturation (location: 10 cm away from the inlet end of the core)

4.2.5 Data acquisition system

The original data acquisition system used by Ayub (2000) and Ayodele (2004) was developed using LabVIEW 4.1 on a PC-Pentium 200 MHz computer. Five years later, to improve the overall efficiency and to increase the flexibility, the data acquisition system was improved by utilizing LabVIEW 7.1 on a DELL compatible PC-Pentium 2.8 GHz computer with Microsoft Windows XP Professional Version 2 as the operating system. Some related interface cards were also changed.

4.2.5.1 Hardware configuration and modifications

The diagram representing the hardware configuration for data collection and instrument control is given in Figure 4-9. This diagram is similar to the one for the old system except for some modifications. Only the changes made are specified here. The details of the unchanged parts can be found in the work by Ayub (2000).

4.2.5.1.1 RS-232 interface card

In the old system, four electronic balances were connected to the RS-232 interface card of the computer via a Byteway (manufactured by Protec Microsystems, Inc.), which is a microprocessor based peripheral sharer. The Byteway was not used in the new data acquisition system because it can slow down the communication between the electronic balances and the computer. Accordingly, the four electronic balances were connected to the data acquisition computer through four RS-232 interface cards, as shown in Figure 4-9.

4.2.5.1.2 E Series Full-Featured Multifunction DAQ

In order to make the stepping motor run more smoothly, the DAS-8 interface card of the old system was replaced in the new system by an E Series Full-Featured Multifunction DAQ card (manufactured by National Instruments Inc.), PCI-MIO-16E-4. The interface between this Multifunction DAQ card and the steeping motor controller is given in Figure 4-10.

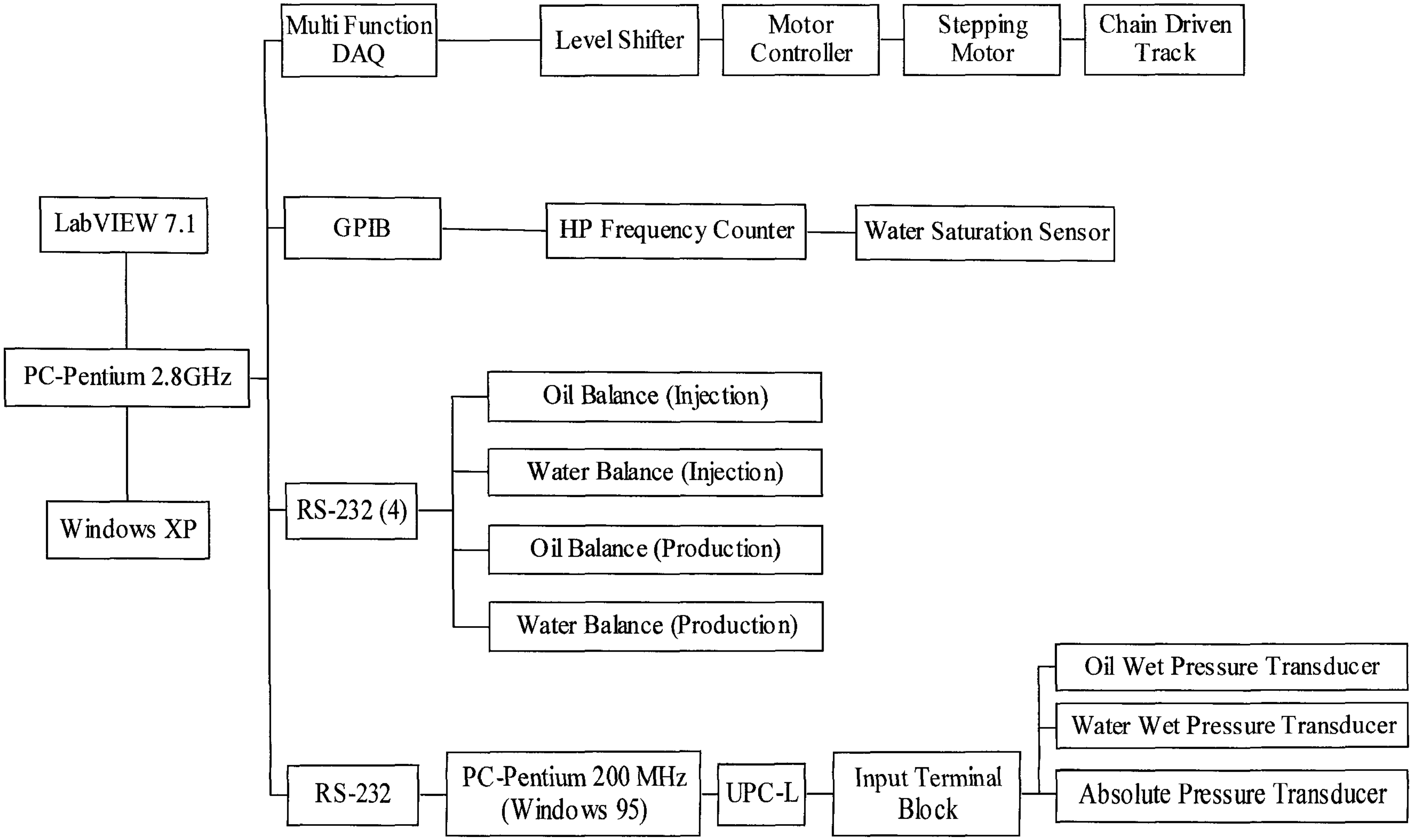


Figure 4-9: Hardware configuration for instrument control and data acquisition

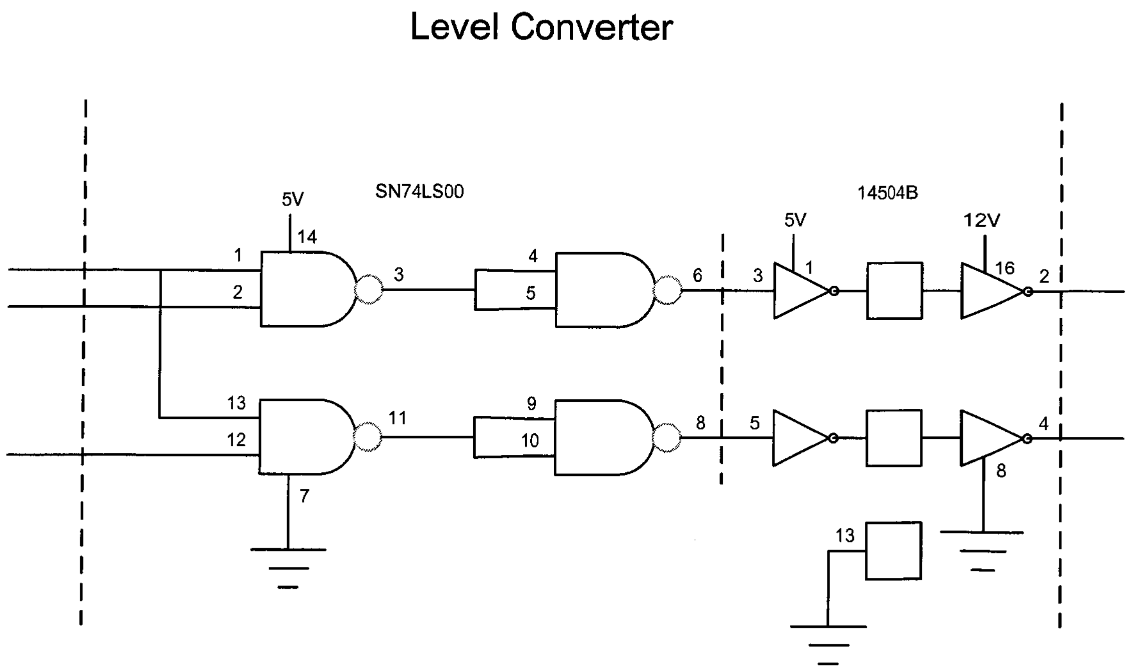


Figure 4-10: Schematic for the interface between Multifunction DAQ card and stepping motor controller

4.2.5.1.3 UPC-L interface card and Easy Sense software

The UPC-L interface card (manufactured by Validyne Engineering Corp.) provides the ability to interface a computer directly to variable reluctance sensors such as Validyne Model DP15 variable reluctance pressure transducers, without using external signal conditioning equipment. Unfortunately, this type of card, currently, just comes with an ISA bus, but not a PCI bus. Therefore, it was decided to run a UPC-L card in the old computer to acquire the pressure data directly from the pressure transducers. By using a special program, the acquired data were then transferred via a RS-232 interface card into the new computer, which contains the main data acquisition program, but does not have an ISA slot to fit the UPC-L card.

In addition, the original configuration software of the UPC-L card, Easy Sense software, which comes with the UPC-L card was not used in the old system, was used in the new system to configure the card.

4.2.5.1.4 GPIB interface card

The general purpose interface bus (GPIB) card (manufactured by National Instruments Inc.) was used to connect the data acquisition computer to the frequency counter (HP Universal Counter 225 MHz, Model 53131-A) which captures the frequency signals from the water measurement sensor.

4.2.5.2 Modifications to the data acquisition software

During the preliminary runs of the experiments, it was found that the tightness of the chain, which was used to the move the core holder when measuring the water saturation, and hence the smoothness of the movement of the core holder, was different when the stepping motor was moving back than when the stepping motor was moving forward. According to the principles underlying the measurement of saturation discussed in Section 4.2.4, the more smoothly the core holder moves, the more accurately the frequency response can reflect the water saturation. The old data acquisition program could acquire only the frequency response data while the water saturation sensor was moving relatively from the outlet towards the inlet end of the core. When moving in this direction, the chain was relatively loose and the core holder moved with more vibration.

In order to find out how much the vibration due to the looseness of the chain affected the accuracy of the water saturation measurements, the old data acquisition program was modified slightly. The new program is able to take frequency data when the core holder is moving in both the forward and the backward directions.

4.2.6 The impact of a relatively loose chain

By using the modified data acquisition program, two sets of frequency data corresponding to the same steady state situation (that is, the same saturation level) were recorded when the core holder was moving backward and forward. These data are plotted in Figure 4-11. In the legend of Figure 4-11, forward refers to the situation when the saturation measurement sensor was moving from the outlet end towards the inlet end of the core; in this direction, the driving chain was relatively loose. Backward refers to the situation when the saturation measurement sensor was moving from the inlet end towards the outlet end of the core; in this direction, the driving chain was relatively tight. As can be seen from Figure 4-11, the backward curve is more horizontal, and hence better, than the forward curve. However, the difference is quite small.

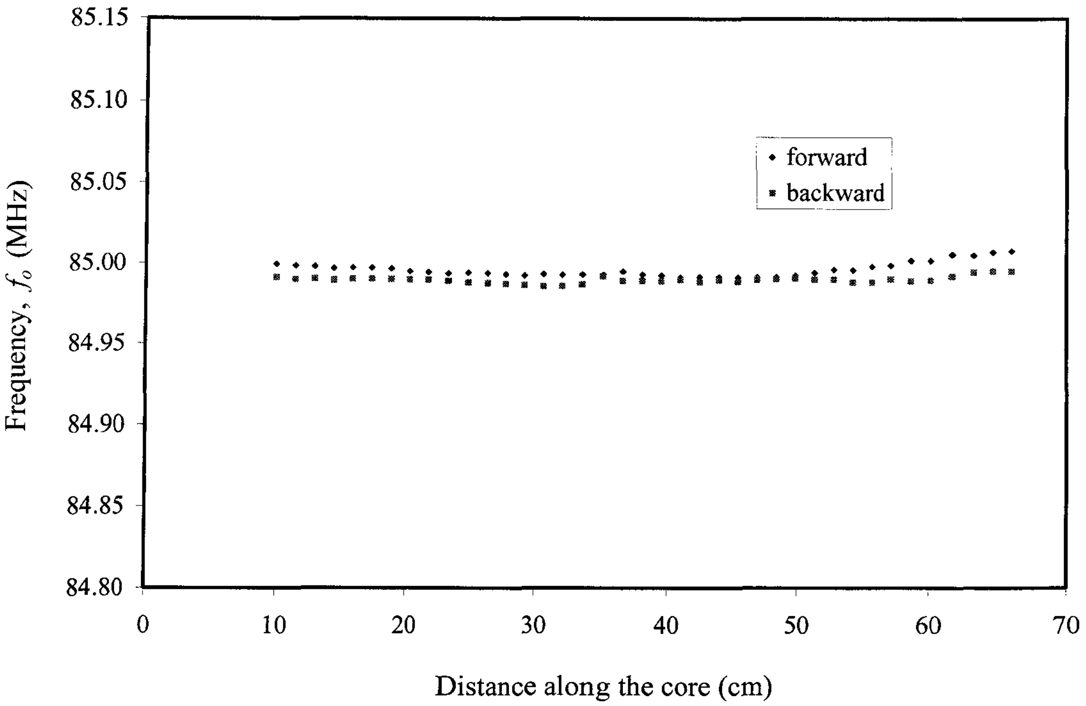


Figure 4-11: Impact of the moving direction on the frequency responses

In this study, because the saturation measurement system had been calibrated using forward curves before the newly improved data acquisition program was available, and the difference between the forward curve and backward curve is not too significant, it was decided not to recalibrate the saturation measurement system by using the backward curves so as to save some time.

4.3 Experimental procedures

The experiments in this study consist mainly of three parts: core preparation, steady state cocurrent (SSCO) flow experiments and unsteady state cocurrent (USCO) flow experiments. The procedures followed to conduct these parts of the experiment and the related discussion are described below.

4.3.1 Core preparation

To obtain a core as uniform as possible, two packing techniques, a wet packing technique and a dry packing technique, were tried in this study.

4.3.1.1 Wet packing and dry packing

In both the wet and the dry packing process, the core holder was hung vertically with a Type B end cap attached at the bottom end and a funnel installed at the top end of the core holder. The funnel used could hold an extra amount of sand when the core holder was full, and a portion of this amount of sand could fill the core holder later on when the sand in the core holder was compacted during the vibration process.

The wet packing technique kept a constant height of water in the core holder hung vertically when sand was poured into it. In the dry packing process, sand was poured into an empty core holder which was hung vertically.

In both techniques, when the core holder was fully filled with sand, an air driven mechanical vibrator was used to vibrate the core holder so as to achieve a fairly compact core.

Because the core holder used in this study was made of Acrylic, and hence was too delicate to tolerate a long time and high frequency vibration, 10 minutes of low frequency vibration was used in the core packing process. In contrast, at least 24 hours of relatively high frequency vibration were usually undertaken in the core packing process used elsewhere. As a consequence, the cores obtained here were not as compact as the ones commonly used in the other experiments.

In addition, it was found that the core obtained by using the wet packing technique had a higher porosity; that is, it was less compact, than that achieved using the dry packing process. Therefore, it was decided to use the dry packing technique to prepare the cores used in the current experiments.

4.3.1.2 Packing procedures followed

1. The core holder was suspended in the vertical position with a Type B end cap attached at the bottom end, and a funnel installed at the top end and dummy plugs inserted into the pressure ports.
2. To avoid the formation of different distinct layers in the sand in the core, the sand was continuously poured into the core holder through the funnel until the core holder and the funnel were full.
3. An air driven mechanical vibrator with a low vibration frequency was used to vibrate the core for about 10 minutes.
4. The funnel was then replaced with a Type A end cap.
5. The dry core holder with the end-caps, fittings and dummy plugs was then weighed.
6. Then water was injected at the bottom end and produced from the top end of the core at a very low rate for at least 24 hours to saturate fully the core and to avoid air entrapment.
7. The core holder was weighed again with the same accessories as mentioned in Step 5.
8. The porosity of the core was calculated by estimating the pore volume with the help of the density of the injected water and the weight values obtained in Step 5 and Step 7.
9. The core was allowed to rest for about 24 hours with 100% water saturation to render it completely water wet.

4.3.2 SSCO flow experiments

There were three purposes for conducting SSCO flow experiments in this study. Firstly, it was necessary to measure the relative permeabilities. Secondly, it was necessary to correlate the frequency responses and the true water saturations of the core. Thirdly, it was necessary to obtain the dynamic capillary pressure curve. It should be noted here that the relative permeabilities and the capillary pressure must be measured for an imbibition process.

The procedures followed to conduct the SSCO flow experiment are given below.

1. The prepared core was placed on the chain driven cart, and then the saturation measurement system was installed. A check was made to make sure that the core could move back and forth horizontally along the chain driven track without any twists, as this ensured the accuracy of the water saturation measured. From this point on, until the whole experiment (including SSCO flow experiment and USCO flow experiment) was finished, changes made in the saturation measurement system, such as the position of the copper conductors, could result in inconsistent saturation measurements.
2. The Type B end cap was then replaced with the Type C end cap.
3. Mineral oil was injected at an appropriate injection rate for at least 8 hours. The pressure limitation of the core holder (25 psi maximum) was kept in mind.
4. The stepping motor and frequency counter were started to let them warm up.
5. At the cessation of water production recorded at the outlet end of the core, the oil wet transducers were prepared by fully filling their corresponding cavities with mineral oil. Then the dummy plugs on one side of the core holder were replaced with the prepared oil wet transducers by adding additional oil to the oil pressure ports of the core holder to avoid any air entrapment.
6. Next oil was injected at the same rate until steady state was achieved. (Attainment of steady state conditions can be confirmed by scanning the core to

see if the frequency responses along the core are almost the same or not.) Then the core was scanned to record the frequency responses.

7. Next the weights of the fluid injected and the fluid produced were recorded to calculate the irreducible water saturation level using the material balance method.

8. Then the effective permeability to the oil at the irreducible water saturation was obtained by using Darcy's law.

9. Next the Type C end cap at the inlet end of the core was replaced with a Type A end cap. Care was taken to make sure that no air was trapped during this procedure.

10. Then oil and water were injected simultaneously, at a certain water-oil ratio, until steady state was achieved.

11. Next the water wet transducers were prepared by filling fully their corresponding cavities with water. Then the dummy plugs on the other side of the core holder were replaced with the prepared water wet transducers by adding additional water to the water pressure ports of the core holder to avoid any air entrapment.

12. Then oil and water were injected simultaneously at the same water-oil ratio as used in Step 10 until steady state was achieved. The core was scanned to record the frequency responses.

13. Then the weights of the fluid injected and the fluid produced were recorded to calculate the corresponding water saturation level using the material balance method.

14. Next the effective permeabilities to both the oil and the water were calculated at the saturation level obtained in Step 13.

15. Then the water wet transducers were replaced with the dummy plugs to prevent the water wet membranes on the water transducers from being contaminated.

16. Finally Steps 10 through 15 were repeated with an increased water-oil ratio until data corresponding to various water saturation levels had been obtained. It should

be noted that the frequency responses and effective permeabilities corresponding to the residual oil saturation have to be measured, too.

4.3.3 USCO flow experiments

Once the relative permeability curve, the capillary pressure curve and the saturation frequency relationship were obtained from the SSCO flow experiment, the USCO flow experiment was conducted on the same core to acquire the saturation profiles (from frequency profiles) and pressure profiles relating to certain injection rate and time points. These measured data were used to compare with the simulation results presented in Chapter 6.

The procedures followed to conduct the USCO flow experiment are presented below.

1. It was verified that a Type C end cap was installed at the inlet end of the core.
2. Then oil was injected until the irreducible water saturation was obtained. Whether the irreducible water saturation condition has been obtained can be confirmed by checking the produced fluid at the outlet end of the core and by checking the frequency responses.
3. Next the Type C end cap was replaced with the Type B end cap at the inlet end of the core.
4. Then the core was scanned to get the frequency response corresponding to the irreducible water saturation. Injection of water was started.
5. Then the core was scanned at regular time intervals to determine the saturations, the phase pressures, and the weight data for further analysis.
6. Every USCO flow experiment run was stopped after obtaining a couple of scans, once the flood front (indicated by the foot of the frequency profile) reached a point 66 cm away from the inlet end of the core.

4.4 Experimental results

In this study, three cores were packed with 80-120 mesh Ottawa silica sand. The first core was packed by using the wet packing technique, while the last two were packed using the dry packing technique. The first two cores were used to conduct preliminary two-phase flow experiments. The last one was used in the final experiment which contains one set of SSCO flow data and two sets of USCO flow data. The fluid phases used in both the preliminary and the final experiments were water and mineral oil.

4.4.1 Results of preliminary experiments

The purposes of the preliminary experiments were: (a) to get familiar with the experimental equipment and the experimental procedures; (b) to test the effectiveness of various water wet membranes for sensing the water phase pressure; and (c) to observe the fluid distribution and the propagation of the front by dyeing the mineral oil purple.

Several conclusions can be drawn on the basis of these preliminary experimental results.

1. The core prepared using the wet packing technique had a higher porosity than that packed using the dry packing technique, provided that the core was vibrated at a low frequency for a relatively short time (10 minutes in the current experiments).
2. The horizontalness of the core, which determined the goodness of the contact between the central conductor and the core, had a great impact on the frequency responses of the water saturation measurement sensor. (Consequently, the chain driven trolley was shortened to improve the horizontalness of the core holder.)
3. The combination of a MF-Millipore hydrophilic membrane (MF disc Mixed Cellulose Esters, provided by Millipore Corporation) and a high porosity support base provided a high permeability to water and a high threshold pressure to

mineral oil and, hence, gave an acceptable, but not perfect, ability to sense the water phase pressure.

4. Neither channeling nor viscous fingering was observed in experiments using cores packed using either the wet packing or the dry packing technique.

5. The pressure ports for the water pressure transducers were apt to be surrounded by the oil phase, which could be attributed to the fact that installation and un-installation of the water pressure transducers might have destroyed the local structure of the core, resulting in the creation of large pores.

4.4.2 Results of the final experiments

To see the repeatability of the experiment, one SSCO flow experiment (Run 5) and two USCO flow experiments (Runs 6 and 7) were conducted on the same core with the same fluids phases (Mineral oil and water). The properties of the core and the fluids are given in Table 4-1.

Core measurement length	0.66 m
Core width	0.05 m
Core height	0.01 m
Core inclination	0 Degree
Sand type	Ottawa silicate sand (80-120 mesh)
Average porosity	0.313
Wetting phase	Water
Nonwetting phase	Mineral Oil
Wetting phase density at room temperature	994 Kg/m^3
Wetting phase viscosity at room temperature	0.001 Pa.s
Nonwetting phase density at room temperature	845 Kg/m^3
Nonwetting phase viscosity at room temperature	0.038 Pa.s
Flow type	SSCO, USCO
Initial wetting phase saturation S_{1i}	0.10
Residual non-wetting phase saturation S_{2r}	0.12
Cocurrent effective perm. to non-wetting phase at S_{1i}	8.721E-12 m^2
Cocurrent effective perm. to wetting phase at S_{2r}	1.511E-12 m^2

Table 4-1: Properties of Core and Fluids

Runs 5 and 6 were conducted using the old data acquisition system and Run 7 was realized using the new data acquisition system.

4.4.2.1 SSCO flow experimental results

By following the procedures outlined in Section 4.3.2, dynamic capillary pressure data, relative permeability data and saturation calibration (saturation-frequency relationship) data were obtained for 8 different saturation levels, from irreducible water saturation to residual oil saturation.

4.4.2.1.1 Dynamic capillary pressure data

Dynamic capillary pressures for various water saturation levels were calculated using the fluid phase pressures acquired from the two pressure transducers which were located closest to the inlet end of the core. These data are tabulated in Table 4-2 and are also plotted in Figure 4-12. (The steady state pressure profiles measured for different water saturation levels are given in Appendix B)

S_l	0.10	0.20	0.28	0.40	0.47	0.57	0.73	0.88
Normalized S	0.000	0.128	0.231	0.385	0.474	0.603	0.808	1.000
P_c (Pa)	4176	3611	3280	2890	2650	2380	2265	1800

Table 4-2: Dynamic capillary pressure

It should be noted here that the capillary pressure is not zero at the residual oil saturation as is usually observed in imbibition processes. This phenomenon can be accounted for by the following reasons.

1. The capillary pressure data shown here were calculated from the dynamic phase pressures instead of being measured under static equilibrium conditions, as is usually the case. Moreover, the fluid-fluid interfaces under dynamic situations must be different from those under static equilibrium conditions.

2. The phase pressures acquired were not very accurate due to the difficulties mentioned in Section 4.2.3.

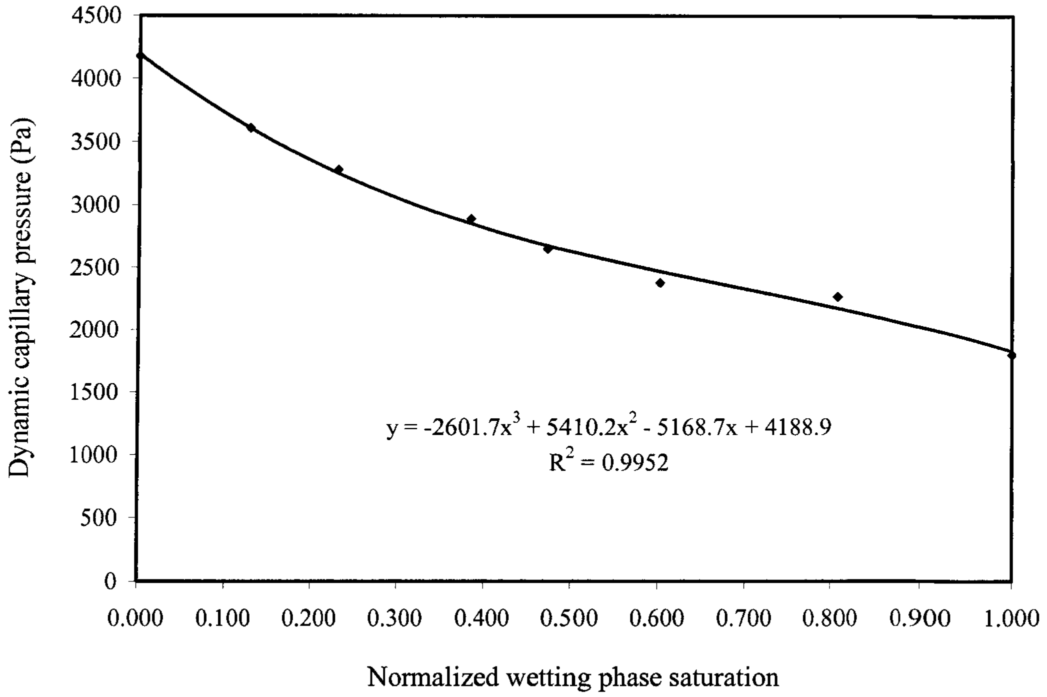


Figure 4-12: Normalized dynamic capillary pressure curve

4.4.2.1.2 Relative permeability data

As can be seen from Equations (3.33) and (3.34), in a horizontal, steady state, co-current flow experiment, the gravitational force and the capillary force disappear and the Modified Transport Equations of two phase flow in porous media degenerate to the traditional Muskat's equations. Therefore, one may use Muskat's equations to calculate the relative permeabilities in a horizontal SSCO flow experiment and then use them in the Modified Transport Equations for USCO flow situations.

Note that, in order to be consistent with the definition of relative permeability used in the simulator developed in Charter 5, the relative permeabilities presented below were defined in terms of the end point permeability. The shapes of the relative permeability curves in the normalized water phase saturation domain are shown in Figure 4-13. The detailed data for relative permeability calculation can be found in Appendix C.

4.4.2.1.3 Saturation calibration data

The frequency responses corresponding to different saturation levels are shown in Figure 4-7. As discussed in Section 4.2.4.3, the frequency responses along the core are not constant values. To reduce the noise related to the measurement positions, the frequency was calibrated in terms of the saturation, and also the measurement positions, by using 39 equations which are associated with 39 different locations along the core. The coefficients are presented in Appendix A.

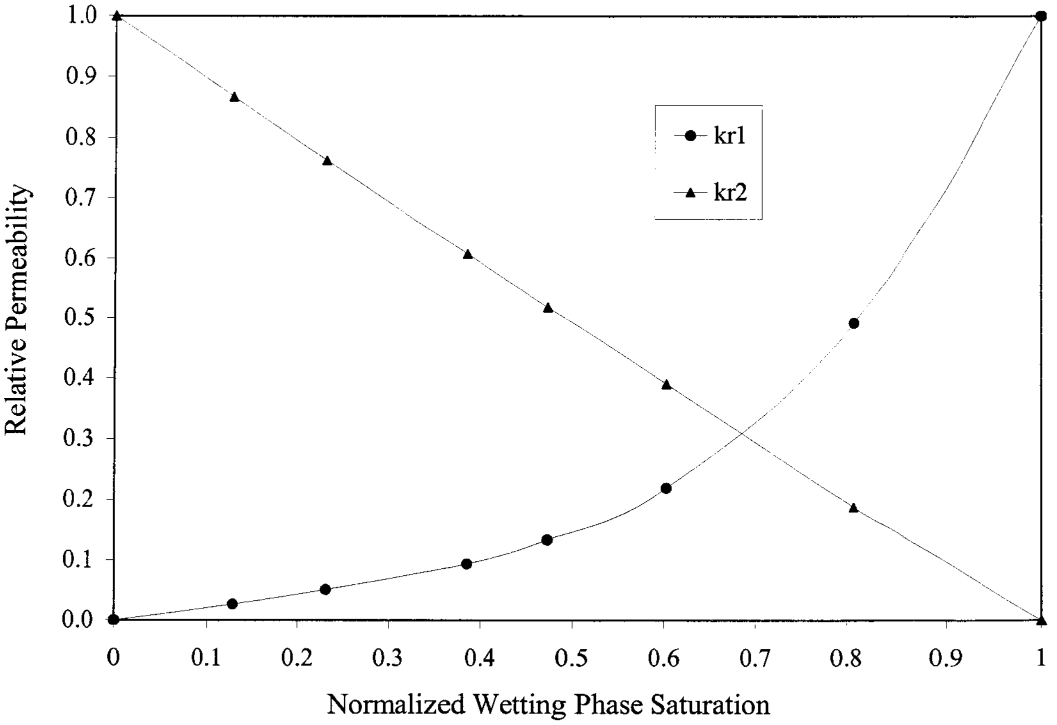


Figure 4-13: Normalized relative permeability curves

4.4.2.2 USCO flow experimental results

The USCO flow data can be depicted by the saturation and pressure profiles associated with different times at a certain injection rate. In the current experiments, by following the procedures described in Section 4.3.3, two USCO flow experiments, Runs 6 and 7, were conducted with water injection rates of 2.201E-8 m^3/s and 3.432E-8 m^3/s, respectively. The saturation and pressure profiles associated with different times were recorded and are presented in Sections 4.4.2.2.1 and 4.4.2.2.2.

4.4.2.2.1 Saturation profiles in USCO flow experiments

The raw frequency profiles for Runs 6 and 7 are shown in Figures 4-14 and 4-15, respectively. With the help of the saturation-frequency calibration equations obtained in the SSCO flow experiment (Run 5), the raw frequency profiles of Runs 6 and 7 can

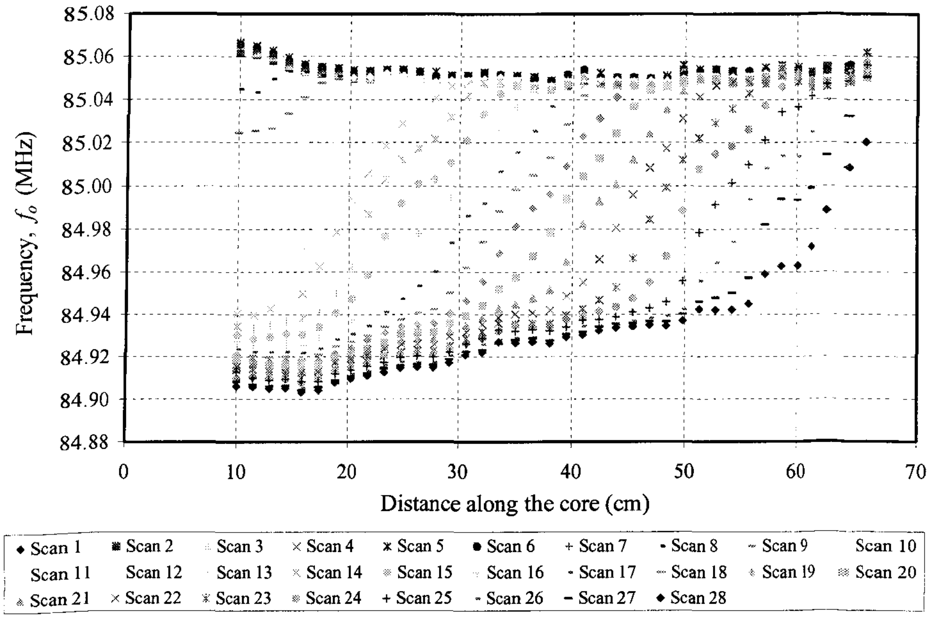


Figure 4-14: Frequency profiles measured from USCO flow experiment Run 6

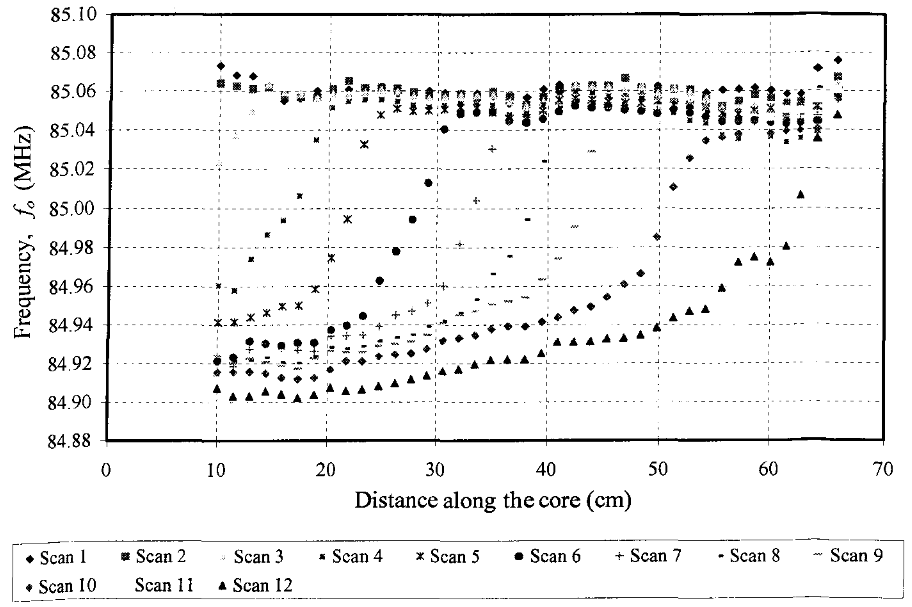


Figure 4-15: Frequency profiles measured from USCO flow experiment Run 7

be converted into the corresponding normalized saturation profiles as shown in Figures 4-16 and 4-17.

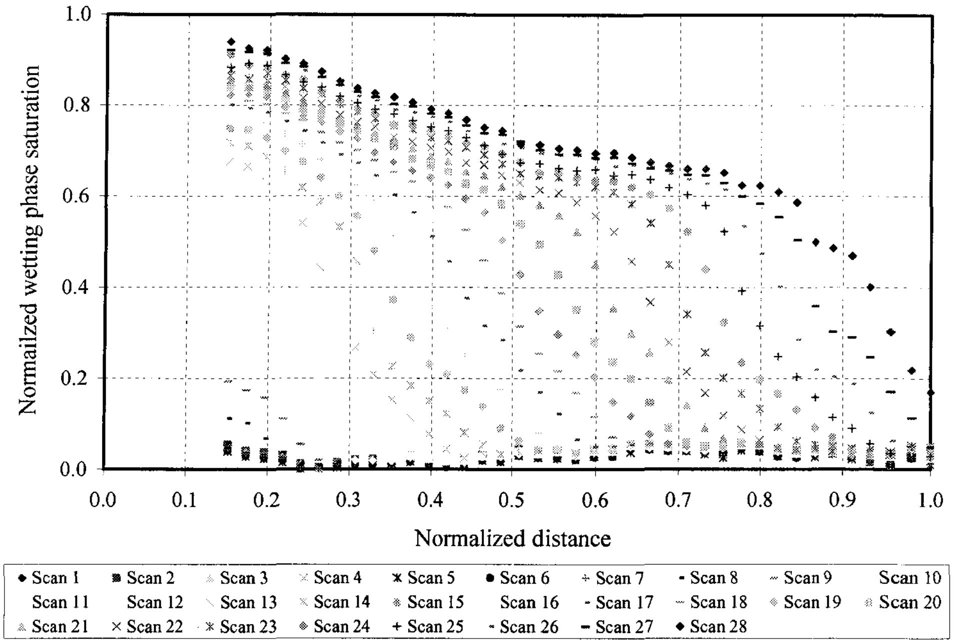


Figure 4-16: Normalized wetting phase saturation profiles for USCO flow experiment Run 6

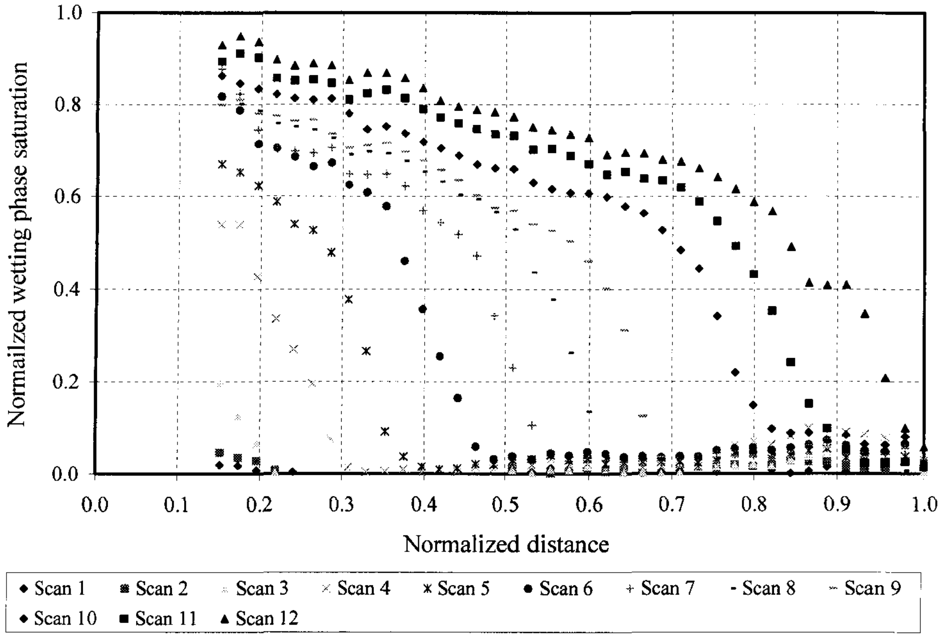


Figure 4-17: Normalized wetting phase saturation profiles for USCO flow experiment Run 7

4.4.2.2.2 Pressure profiles in the USCO flow experiments

The pressure profiles for the USCO experiments contain a large amount of data; consequently, only the pressure profiles corresponding to Scan 27 of Run 6 and Scan 12 of Run 7, which are used in the simulation analysis in Chapter 6, are presented here (see Figures 4-18 and 4-19). The rest of the data is available in the CD located at the back of the thesis.

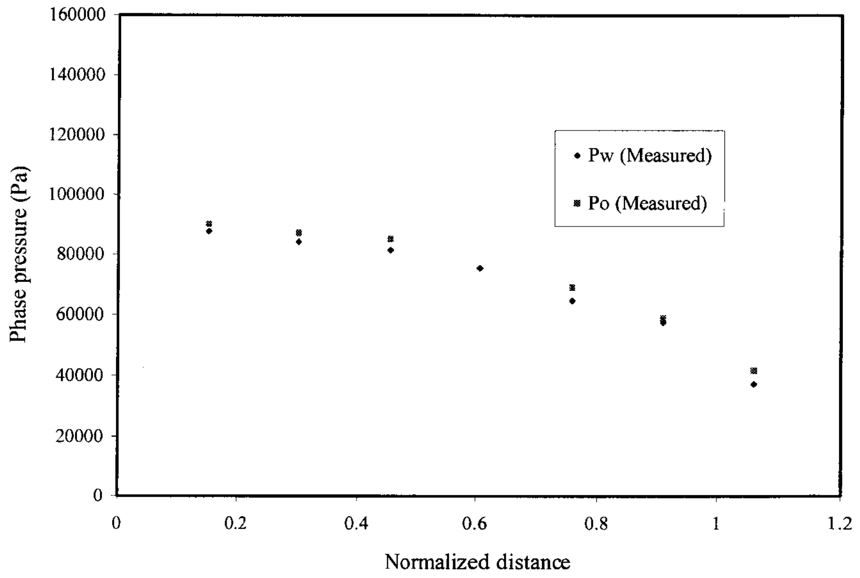


Figure 4-18: Pressure profiles for Scan 27 of USCO flow experiment Run 6

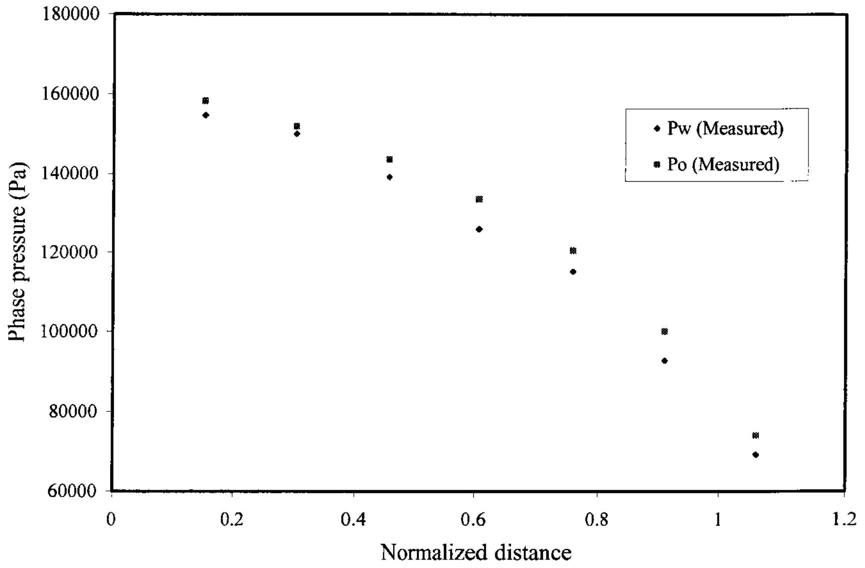


Figure 4-19: Pressure profiles for Scan 12 of USCO flow experiment Run 7

CHAPTER 5

Numerical simulator development and validation

5.1 Introduction

In order to determine the ability of the Modified Transport Equations constructed in Chapter 3 to describe the phenomena of one dimensional cocurrent flow of two incompressible and immiscible fluids through a homogenous and isotropic porous medium, a numerical simulator based on these equations was developed. This was achieved by utilizing the fractional flow concept, and Bentsen's equation, a Lagrangian form of the flow equations derived by Bentsen (1978). In this simulator, the variable inlet saturation effect was taken into account by using the material balance check technique proposed by Shen and Ruth (1994). Moreover, to ensure that the simulator worked properly when handling the variable inlet saturation effect, the simulator was validated theoretically and experimentally.

5.2 Theory and background

As is known, the one dimensional, two phase, immiscible displacement process can be described mathematically by six equations, together with some initial and boundary conditions. These six equations are: two transport equations (the Modified Transport Equations are used here),

$$v_1 = -\lambda_1^0 \left\{ \frac{\partial \psi_1}{\partial x} + \left(\frac{1-\alpha_1}{2} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \right] \right\} , \tag{5.1}$$

and

$$v_2 = -\lambda_2^0 \left\{ \frac{\partial \psi_2}{\partial x} - \left(\frac{1-\alpha_2}{2R_{12}} \right) \left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \right] \right\} ; \tag{5.2}$$

two continuity equations for the two incompressible fluids,

$$\frac{\partial v_1}{\partial x}+\phi\frac{\partial S_1}{\partial t}=0\,, \tag{5.3}$$

and

$$\frac{\partial v_2}{\partial x}+\phi\frac{\partial S_2}{\partial t}=0\,; \tag{5.4}$$

the equation relating the capillary pressure gradient and the phase potential gradients of the two fluid phases,

$$R_{12}\frac{\partial \psi_2}{\partial x}-\frac{\partial \psi_1}{\partial x}=\frac{\partial P_c}{\partial x}-(\rho_1-\rho_2 R_{12})g\sin\theta\,; \tag{5.5}$$

and the equation linking the saturations of the two fluid phases in the porous medium,

$$S_1+S_2=1\,. \tag{5.6}$$

The direct combination of these six equations results in a second order, nonlinear, parabolic, partial differential equation in Eulerian form. One can solve numerically this partial differential equation by discretizing it with respect to its dependent variables x and t, as is usually done in most commercial simulators. The drawback of this approach is that it is hard to handle the variable inlet saturation effect.

Equations (5.1) through (5.6) may be combined in another way. In particular, by utilizing the fractional flow concept [Leverett (1941)] and the frontal advance equation [Buckley and Leverett (1942)], Bentsen (1978) derived a Lagrangian equation, known as Bentsen's equation. Bentsen's equation includes all the major governing factors in two phase flow through porous media, and its use leads to a very simple numerical scheme to analyze the displacement process. Moreover, because Bentsen's equation is a Lagrangian form equation, one can discretize it in the saturation domain and, in turn, deal with the variable inlet saturation effect easily. Therefore, Bentsen's equation is employed to develop the numerical simulator used in this study.

The numerical simulator developed in this study resembles the one developed in the thesis by Ayodele (2004). The differences between these two simulators are mainly in the following aspects:

1. the algorithm for coding,
2. discretization of Bentsen's equation,
3. initial and boundary conditions,
4. the way of handling the variable inlet saturation effect,
5. the Newton-Raphson solver,
6. the way of implementing the JAVA$^{\text{TM}}$ method.

The simulator development process is given in the following sections.

5.3 Mathematical Formulation

In this section, Equations (5.1) through (5.6) are manipulated to obtain the fractional flow equation and the frontal advance equation. These two equations are then normalized by employing the normalization technique used by Bentsen (1976, 1978), Shen and Ruth (1994) and Ayodele (2004). Finally, Bentsen's equation is derived based on the normalized fractional flow equation and normalized frontal advance equation.

5.3.1 Fractional flow equation

One can obtain the fractional flow equation, using the Modified Transport Equations, by applying the fractional flow concept proposed by [Leverett (1941)] and by rearranging Equations (5.1), (5.2) and (5.5) in the following manner.

Firstly, solving Equations (5.1) and (5.2) for $\frac{\partial \psi_1}{\partial x}$ and $R_{12}\frac{\partial \psi_2}{\partial x}$ leads to:

$$\frac{\partial \psi_1}{\partial x} = -\frac{v_1}{\lambda_1^0} - \left(\frac{1-\alpha_1}{2}\right)\left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12}) g \sin\theta\right] \tag{5.7}$$

and

$$R_{12}\frac{\partial \psi_2}{\partial x} = -\frac{R_{12}v_2}{\lambda_2^0} + \left(\frac{1-\alpha_2}{2}\right)\left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12})g\sin\theta\right]. \tag{5.8}$$

Subtracting Equation (5.7) from Equation (5.8), one gets:

$$R_{12}\frac{\partial \psi_2}{\partial x} - \frac{\partial \psi_1}{\partial x} = -\frac{R_{12}v_2}{\lambda_2^0} + \frac{v_1}{\lambda_1^0} + \left(1 - \frac{\alpha_1+\alpha_2}{2}\right)\left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12})g\sin\theta\right]. \tag{5.9}$$

Subtracting Equation (5.5) from Equation (5.9) and rearranging the resulting equation, one gets:

$$\frac{v_1}{\lambda_1^0} - \frac{R_{12}v_2}{\lambda_2^0} = \left(\frac{\alpha_1+\alpha_2}{2}\right)\left[\frac{\partial P_c}{\partial x} - (\rho_1 - \rho_2 R_{12})g\sin\theta\right]. \tag{5.10}$$

It is known that the total influx, *v*, is the sum of the fluxes of the two flowing phases,

$$v = v_1 + v_2 . \tag{5.11}$$

Combining Equations (5.10) and (5.11) to eliminate the influx of phase two, v_2, and rearranging the resulting equation, one obtains the fractional flow equation:

$$f_1 = \frac{v_1}{v} = F_1\left\{1 + \frac{(\alpha_1+\alpha_2)}{2}\frac{\lambda_2^0}{vR_{12}}\left(\frac{\partial P_c}{\partial x} - (\rho_1 - R_{12}\rho_2)g\sin\theta\right)\right\}, \tag{5.12}$$

where

$$F_1 = \frac{R_{12}\lambda_1^0}{R_{12}\lambda_1^0 + \lambda_2^0} . \tag{5.13}$$

For convenience sake, the defining equations for the related parameters in Equation (5.12) are presented below. The derivation of these defining equations can be found in Chapter 3.

$$R_{12} = 1 - a(1 - S), \tag{5.14}$$

$$\alpha_i = \alpha_{vi}\alpha_{ci} \qquad i=1,2, \tag{5.15}$$

$$\alpha_{v1} = 1 - \frac{c}{R_{12}}\frac{\lambda_2^0}{\lambda_m^0}, \tag{5.16}$$

$$\alpha_{v2} = 1 - cR_{12}\frac{\lambda_1^0}{\lambda_m^0}, \tag{5.17}$$

$$\alpha_{c1} = \alpha_{c2} = \alpha_c = 1 - \phi, \tag{5.18}$$

and

$$\lambda_m^0 = S\lambda_{1r}^0 + (1 - S)\lambda_{2r}^0. \tag{5.19}$$

5.3.2 Frontal advance equation

By combing Equations (5.3), (5.4) and (5.6), one can get the frontal advance equation presented below. Adding Equations (5.3) and (5.4), and taking into account Equations (5.6) and (5.11), one obtains:

$$\frac{\partial v}{\partial x} = 0. \tag{5.20}$$

Then, replacing v_1 by $f_1 v$ in Equation (5.3) and combining the result with Equation (5.20) yields:

$$\frac{\partial f_1}{\partial x} = -\frac{\phi}{v}\frac{\partial S_1}{\partial t}. \tag{5.21}$$

Because f_1 is a function of wetting phase saturation S_1, Equation (5.21) can be written in the following form:

$$\frac{\partial f_1}{\partial S_1}\frac{\partial S_1}{\partial x} = -\frac{\phi}{v}\frac{\partial S_1}{\partial t}. \tag{5.22}$$

Moreover, because the wetting phase saturation S_1 is a function of x and t, that is, $S_1(x,t)$, one can write the following expression for saturation change:

$$dS_1 = \frac{\partial S_1}{\partial x}dx + \frac{\partial S_1}{\partial t}dt. \tag{5.23}$$

For a fluid front of constant saturation during the displacement process, Equation (5.23) becomes:

$$\frac{\partial S_1}{\partial t} = -\frac{\partial S_1}{\partial x}\frac{\partial x}{\partial t}. \tag{5.24}$$

Substituting Equation (5.24) into Equation (5.22) leads to the following form of the frontal advance equation:

$$\frac{\partial x}{\partial t} = \frac{v}{\phi}\frac{\partial f_1}{\partial S_1}. \tag{5.25}$$

5.3.3 Normalization of the fractional flow equation and frontal advance equation

In order to handle the equations easily and to analyze the effects of various parameter groups, the fractional flow equation [Equation (5.12)] and the frontal advance equation [Equation (5.25)] are normalized after the manner of Bentsen (1976, 1978), Shen and Ruth (1994) and Ayodele (2004). The normalized fractional flow equation and frontal advance equation can be written as:

$$f(S,\tau) = G(S) - N_c C(S) / \frac{\partial \xi}{\partial S}, \tag{5.26}$$

and

$$\frac{\partial \xi}{\partial \tau} = \frac{\partial f}{\partial S}, \tag{5.27}$$

where

$$G(S)=\left(1-\frac{\bar{\lambda}(S)\{1+(\rho_2/\Delta\rho')[1-R_{12}(S)]\}N_g k_{r2}(S)}{R_{12}(S)M_r}\right)F_1(S), \tag{5.28}$$

$$N_c=\frac{A_c\lambda_{1r}^0}{vL}\ , \tag{5.29}$$

$$C(S)=-\frac{1}{M_r R_{12}(S)}\bar{\lambda}(S)F_1(S)k_{r2}(S)\frac{d\pi_c}{dS}, \tag{5.30}$$

$$S=\frac{S_1-S_{1i}}{1-S_{2r}-S_{1i}}, \tag{5.31}$$

$$\xi=\frac{x}{L}\ , \tag{5.32}$$

$$\tau=\frac{vt}{\phi L(1-S_{2r}-S_{1i})}, \tag{5.33}$$

$$\bar{\lambda}(S)=\frac{\alpha_1+\alpha_2}{2}, \tag{5.34}$$

$$\Delta\rho'=\rho_1-\rho_2, \tag{5.35}$$

$$N_g=\frac{\lambda_{1r}^0\Delta\rho' g\sin(\theta)}{v}, \tag{5.36}$$

$$M_r=\frac{k_{1r}}{\mu_1}\cdot\frac{\mu_2}{K_{2r}}=\frac{\lambda_{1r}^0}{\lambda_{2r}^0}, \tag{5.37}$$

$$F_1(S)=\frac{R_{12}(S)M_r k_{r1}(S)}{R_{12}(S)M_r k_{r1}(S)+k_{r2}(S)}, \tag{5.38}$$

and

$$\pi_c(S)=\frac{P_c-P_d^{\prime}}{A_c}. \tag{5.39}$$

It should be noted that, for convenience sake, the wetting phase fractional flow symbol, f_1, is replaced by f in the normalized equations [Equations (5.26) and (5.27)].

5.3.4 Bentsen's equation

As can be seen from the derivation procedures presented above, the normalized fractional flow equation and the normalized frontal advance equation are obtained completely from Equations (5.1) through (5.6). Therefore, the normalized fractional flow equation and the normalized frontal advance equation can be used together to describe the one dimensional, two phase, immiscible displacement process through porous media. That is, they are completely equivalent to Equations (5.1) through (5.6).

Bentsen (1978) combined Equations (5.26) and (5.27) in a novel way and eliminated the dependent variable ξ to obtain the Lagrangian form of the immiscible displacement equation, known as Bentsen's equation. Shen and Ruth (1996) derived Bentsen's equation using a slightly different approach. This approach is followed in this study to get Bentsen's equation.

Firstly, rearranging Equation (5.26) yields:

$$\frac{\partial \xi}{\partial S} = -\frac{N_c C(S)}{f(S,\tau) - G(S)} . \tag{5.40}$$

Differentiating Equation (5.40) with respect to τ leads to:

$$\frac{\partial^2 \xi}{\partial S \partial \tau} = \frac{N_c C(S)}{\left[f(S,\tau) - G(S)\right]^2} \frac{\partial f}{\partial \tau} . \tag{5.41}$$

Then, differentiating Equation (5.27) with respect to S, one obtains:

$$\frac{\partial^2 \xi}{\partial \tau \partial S} = \frac{\partial^2 f}{\partial S^2} . \tag{5.42}$$

Finally, substituting Equation (5.41) into Equation (5.42) and rearranging the resulting equation, one gets the following form of Bentsen's equation:

$$[f - G(S)]^2 \frac{\partial^2 f}{\partial S^2} = N_c C(S) \frac{\partial f}{\partial \tau}, \tag{5.43}$$

Employing Bentsen's equation, one can analyze easily the immiscible displacement process. Firstly, one can solve numerically Bentsen's equation to obtain the fractional flow profiles, $f(S, \tau)$. Then, the fractional flow profiles can be used together with Equation (5.27) or Equation (5.40) to get the corresponding saturation profiles, $\xi(S, \tau)$. Finally, once the fractional flow profiles and saturation profiles are known, one can get easily the phase potential gradients by solving the following two equations [Equations (5.44) and (5.45)]:

$$\frac{\partial \psi_1}{\partial \xi} = L\left\{ -\frac{fv}{\lambda_1^0} - \left(\frac{1-\alpha_1}{2}\right)\left[\frac{\partial P_c}{\partial S}\frac{\partial S}{\partial \xi}\frac{1}{L} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \Big]\Big]\right\} \tag{5.44}$$

and

$$\frac{\partial \psi_2}{\partial \xi} = L\left\{ -\frac{(1-f)v}{\lambda_2^0} + \left(\frac{1-\alpha_2}{2R_{12}}\right)\left[\frac{\partial P_c}{\partial S}\frac{\partial S}{\partial \xi}\frac{1}{L} - (\rho_1 - \rho_2 R_{12}) g \sin\theta \Big]\Big]\right\}. \tag{5.45}$$

5.4 Solving Bentsen's equation

Because Bentsen's equation was derived from Equations (5.26) and (5.27), the solution of it has to satisfy these two governing equations. This requirement can be met only by using a variable inlet saturation that guarantees a non-zero $\partial\xi/\partial S$ which is a requirement in the derivation of Bentsen's equation. Moreover, variable inlet saturations are consistent with experimental observations [Jones-Parra et. al. (1954), Kyte and Rapoport (1958)]. With this in mind, Shen and Ruth (1994) derived the following initial and boundary conditions (referred to, in this study, as the Variable Inlet Saturation Conditions).

$$S^*(0) = 0, \tag{5.46}$$

$$\xi(S^*, \tau) = 0, \tag{5.47}$$

$$f(0, \tau) = 0, \tag{5.48}$$

$$f(S^*, \tau) = 1, \tag{5.49}$$

and

$$\int_0^{S^*}\int_S^{S^*} \frac{N_c C(s)}{f(s,\tau) - G(s)} ds dS = \tau , \tag{5.50}$$

where S^* is the time dependent inlet saturation that increases from 0 to 1 as injection continues.

5.4.1 Determination of S^*

To solve Bentsen's equation numerically, one has to know the discretization domain S^*, which can be determined by a material balance check. The checking equation is as follows:

$$\int_0^{S^*(\tau)} \xi(S,\tau) dS = \tau . \tag{5.51}$$

In order to solve Equation (5.51), one needs the saturation profile $\xi(S, \tau)$ which can be obtained by solving the fractional flow equation, Equation (5.26), or the frontal advance equation, Equation (5.27), once $f(S,\tau)$ has been obtained from Equation (5.43). Nevertheless, Shen and Ruth (1994) showed that only the fractional flow equation can be used to determine S^*, because the frontal advance equation was derived from the mass conservation condition and therefore Equation (5.51) is always valid regardless of the magnitude of S^*, if the frontal advance equation is solved for $\xi(S, \tau)$.

The procedures used to determine the time dependent S^* are stated below:

At every time step, after obtaining $f(S,\tau)$ ($0 \leq S \leq S^*$) from Bentsen's equation with a presumed S^*, Equation (5.40) is integrated with respect to S from S to S^*, and keeping in mind the boundary condition described by Equation (5.47), one gets:

$$\xi(S,\tau) = \int_S^{S^*} \frac{N_c C(S)}{f(S,\tau) - G(S)} dS \, . \tag{5.52}$$

Integrating Equation (5.52) from 0 to S^*, one gets the calculated injected normalized pore volume, $\tilde{\tau}(S^*)$, shown as Equation (5.53).

$$\tilde{\tau}(S^*) = \int_0^{S^*} \xi(S,\tau) dS = \int_0^{S^*} \int_S^{S^*} \frac{N_c C(S)}{f(S,\tau) - G(S)} dS dS \, . \tag{5.53}$$

If $\tilde{\tau}(S^*)$ is equal to the actual injected normalized pore volume τ that is measured from the experiment, S^* is valid and one can proceed to the next time step; otherwise, a new S^* should be estimated and the above process repeated until $\tilde{\tau}(S^*) = \tau$.

5.5 Discretization scheme and solution algorithm

Contrary to solving the Eulerian form of the equation, one needs to use saturation as the discretized domain to solve Bentsen's equation, the Lagrangian form of the equation. The discretized saturation domain is from 0 to S^* or 1 if S^* has reached its maximum value. Because the wetting phase fractional flow, $f(S,\tau)$, is known at S=0 and S=S^*, as determined by Equations (5.48) and (5.49), it was decided to use a Point Centered Grid discretization scheme in this work.

In the case where the whole saturation domain is divided into m+1 grid blocks, there will be m+2 grid points, 0, S_1, S_2, ..., S_m, and S_{m+1}. Because $f(0,\tau)$=0 and $f(S^*,\tau)$=1, at every time step (that is, at a certain τ), there will be m unknowns which are $f(S_1,\tau)$, $f(S_2,\tau)$, ... and $f(S_m,\tau)$.

5.5.1 Solution of Bentsen's equation, $f(S,\tau)$

By using a finite difference approximation method, Bentsen's equation can be discretized, backward in time and centered in saturation, into the form of Equation (5.54). This is done in the manner utilized in the thesis by Ayodele (2004), but where several changes have been made. As suggested by Ayodele (2004),

$$f_i^{n+1} - f_i^n = (f_i^{n+1} - G_i^{n+1})^2 (f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1})\chi_i^{n+1}, \tag{5.54}$$

where

$$\chi_i^{n+1} = \frac{\Delta\tau}{N_c C_i^{n+1} \Delta S^2}.$$

Manipulating Equation (5.54), one gets:

$$D_i^{n+1}(f_{i-1}^{n+1}, f_i^{n+1}, f_{i+1}^{n+1}) = (f_i^{n+1} - G_i^{n+1})^2 (f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1})\chi_i^{n+1} - f_i^{n+1} + f_i^n, \tag{5.55}$$

By applying the Newton iteration method at three different saturation points each time, the following equations can be obtained.

$$\frac{\partial D_i^{n+1}}{\partial f_{i-1}^{n+1}} = (f_i^{n+1} - G_i^{n+1})^2 \chi_i^{n+1}, \tag{5.56}$$

$$\frac{\partial D_i^{n+1}}{\partial f_i^{n+1}} = 2\chi_i^{n+1}(f_i^{n+1} - G_i^{n+1})(f_{i+1}^{n+1} - 3f_i^{n+1} + f_{i-1}^{n+1} + G_i^{n+1}) - 1, \tag{5.57}$$

and

$$\frac{\partial D_i^{n+1}}{\partial f_{i+1}^{n+1}} = (f_i^{n+1} - G_i^{n+1})^2 \chi_i^{n+1}. \tag{5.58}$$

Using J and Δf to denote the coefficients of the Jacobian matrix and the vector of unknowns, respectively, one gets the following equation:

$$J \times \Delta f = D\,, \tag{5.59}$$

where Δf is a vector of Δf_i^{n+1} and D is a vector of D_i^{n+1}. The expanded form of Equation (5.59) is:

$$\begin{pmatrix} \frac{\partial D_1^{n+1}}{\partial f_1^{n+1}} & \frac{\partial D_1^{n+1}}{\partial f_2^{n+1}} & & & \\ \frac{\partial D_2^{n+1}}{\partial f_1^{n+1}} & \frac{\partial D_2^{n+1}}{\partial f_2^{n+1}} & \frac{\partial D_2^{n+1}}{\partial f_3^{n+1}} & & \\ & \frac{\partial D_3^{n+1}}{\partial f_2^{n+1}} & \frac{\partial D_3^{n+1}}{\partial f_3^{n+1}} & \frac{\partial D_3^{n+1}}{\partial f_4^{n+1}} & \\ & & \vdots & \vdots & \vdots \\ & & & \frac{\partial D_m^{n+1}}{\partial f_{m-1}^{n+1}} & \frac{\partial D_m^{n+1}}{\partial f_m^{n+1}} \end{pmatrix} \times \begin{pmatrix} \Delta f_1^{n+1} \\ \Delta f_2^{n+1} \\ \Delta f_3^{n+1} \\ \vdots \\ \Delta f_m^{n+1} \end{pmatrix} = \begin{pmatrix} D_1^{n+1}(f_0^{n+1}, f_1^{n+1}, f_2^{n+1}) \\ D_2^{n+1}(f_1^{n+1}, f_2^{n+1}, f_3^{n+1}) \\ D_3^{n+1}(f_2^{n+1}, f_3^{n+1}, f_4^{n+1}) \\ \vdots \\ D_m^{n+1}(f_{m-1}^{n+1}, f_m^{n+1}, f_{m+1}^{n+1}) \end{pmatrix}. \tag{5.60}$$

The Jacobian matrix in Equation (5.60) is a tri-diagonal matrix and hence Equation (5.60) can be solved easily by employing the "Thomas algorithm" which is coded into a Java Method called the "Newton-Raphson Solver" in the simulator. Once the equations are solved, one can get the vector of Δf_i^{n+1} which is defined by the following equations:

$$\Delta f_i^{n+1} = f_i^{n+1} - f_{is}^{n+1} \tag{5.61}$$

and

$$f_{is}^{n+1} = f_i^{n+1} - \Delta f_i^{n+1}\,. \tag{5.62}$$

The variable f_i^{n+1} is the initially assumed or previous iteration value of fractional flow at time step n+1 and grid point i. The variable f_{is}^{n+1} is the true solution of fractional flow at time step n+1 and grid point i.

Now, looking at Equations (5.55) and (5.57), it can be seen that the right hand side (RHS) of Equation (5.57) contains all the unknowns on the RHS of Equation (5.55).

Therefore, it is possible to solve Equation (5.55) by only manipulating Equation (5.57) instead of using all three equations, Equations (5.55), (5.57) and (5.58). In this way, it is possible to apply the Newton iteration method at one saturation point each time; moreover, this approach can be described mathematically by the following equation:

$$\begin{pmatrix} \frac{\partial D_1^{n+1}}{\partial f_1^{n+1}} & & & & \\ & \frac{\partial D_2^{n+1}}{\partial f_2^{n+1}} & & & \\ & & \frac{\partial D_3^{n+1}}{\partial f_3^{n+1}} & & \\ & & & \vdots & \\ & & & & \frac{\partial D_m^{n+1}}{\partial f_m^{n+1}} \end{pmatrix} \times \begin{pmatrix} \Delta f_1^{n+1} \\ \Delta f_2^{n+1} \\ \Delta f_3^{n+1} \\ \vdots \\ \Delta f_m^{n+1} \end{pmatrix} = \begin{pmatrix} D_1^{n+1}(f_0^{n+1}, f_1^{n+1}, f_2^{n+1}) \\ D_2^{n+1}(f_1^{n+1}, f_2^{n+1}, f_3^{n+1}) \\ D_3^{n+1}(f_2^{n+1}, f_3^{n+1}, f_4^{n+1}) \\ \vdots \\ D_m^{n+1}(f_{m-1}^{n+1}, f_m^{n+1}, f_{m+1}^{n+1}) \end{pmatrix}. \quad (5.63)$$

As compared to Equation (5.60), the matrix coefficients of Equation (5.63) do not have the upper and lower diagonal elements. Nevertheless, because $\frac{\partial D_i^{n+1}}{\partial f_i^{n+1}}$ is determined by the properties of the three adjacent saturation points, i-1, i and i+1, the final iteration results should be the same as those obtained from Equation (5.60). This is validated in Section 5.7.

In terms of computation technique, Equation (5.63), due to its simpler form, is easier to solve than Equation (5.60). However, as is turns out, Equation (5.63) requires more iteration steps and computation time than does Equation (5.60). The solution process of Equation (5.63) in the simulator is undertaken by utilizing a Java method called the "Newton-Jacobi solver" in the thesis by Ayodele (2004).

Because the approximation scheme used to obtain Equation (5.54) is an implicit finite difference approximation scheme, which is unconditionally stable, there should be no restriction on the selection of the value of $\frac{\Delta\tau}{\Delta S^2}$ no matter what solution technique, Newton-Raphson Solver or Newton-Jacobi solver, is used. This can be tested by using the simulator developed in this study.

5.5.2 Solution of *ξ*(*S*, *τ*)

As stated in Section 5.4.1, for $S^{*}<1$, one needs to solve the fractional flow equation [Equation (5.26)] for $\xi(S, \tau)$. For $S^{*}=1$, one can use the fractional flow equation [Equation (5.26)] or the frontal advance equation [Equation (5.27)] to get $\xi(S, \tau)$. Equation (5.27) was employed in the simulator for $S^{*}=1$.

The frontal advance equation was discretized as shown below by using the Crank-Nicholson method [Peaceman (1977)].

$$\frac{\xi_i^{n+1}-\xi_i^{n}}{\Delta\tau}=\frac{f_i^{n}-f_{i-1}^{n}}{2\Delta S}+\frac{f_i^{n+1}-f_{i-1}^{n+1}}{2\Delta S} \tag{5.63}$$

Equation (5.63) can be transformed into Equation (5.64) to obtain the distance traveled.

$$\xi_i^{n+1}=\xi_i^{n}+\frac{\Delta\tau}{2\Delta S}\left(f_i^{n}-f_{i-1}^{n}+f_i^{n+1}-f_{i-1}^{n+1}\right) \tag{5.64}$$

5.6 Code development

Based on the algorithm described above, and Ayodele's old version of the Interfacial Coupling Simulator [Ayodele (2004)], referred to as the Interfacial Coupling Simulator (1.0), the new version of the simulator, referred to as the Interfacial Coupling Simulator (2.0), was developed by applying completely the Variable Inlet Saturation Conditions, and by using the Material Balance Check method. For the sake of convenience, the Java$^{\text{TM}}$ programming language was still used in the development of the Interfacial Coupling Simulator (2.0). The flow chart of the new simulator is presented in Figure 5-1.

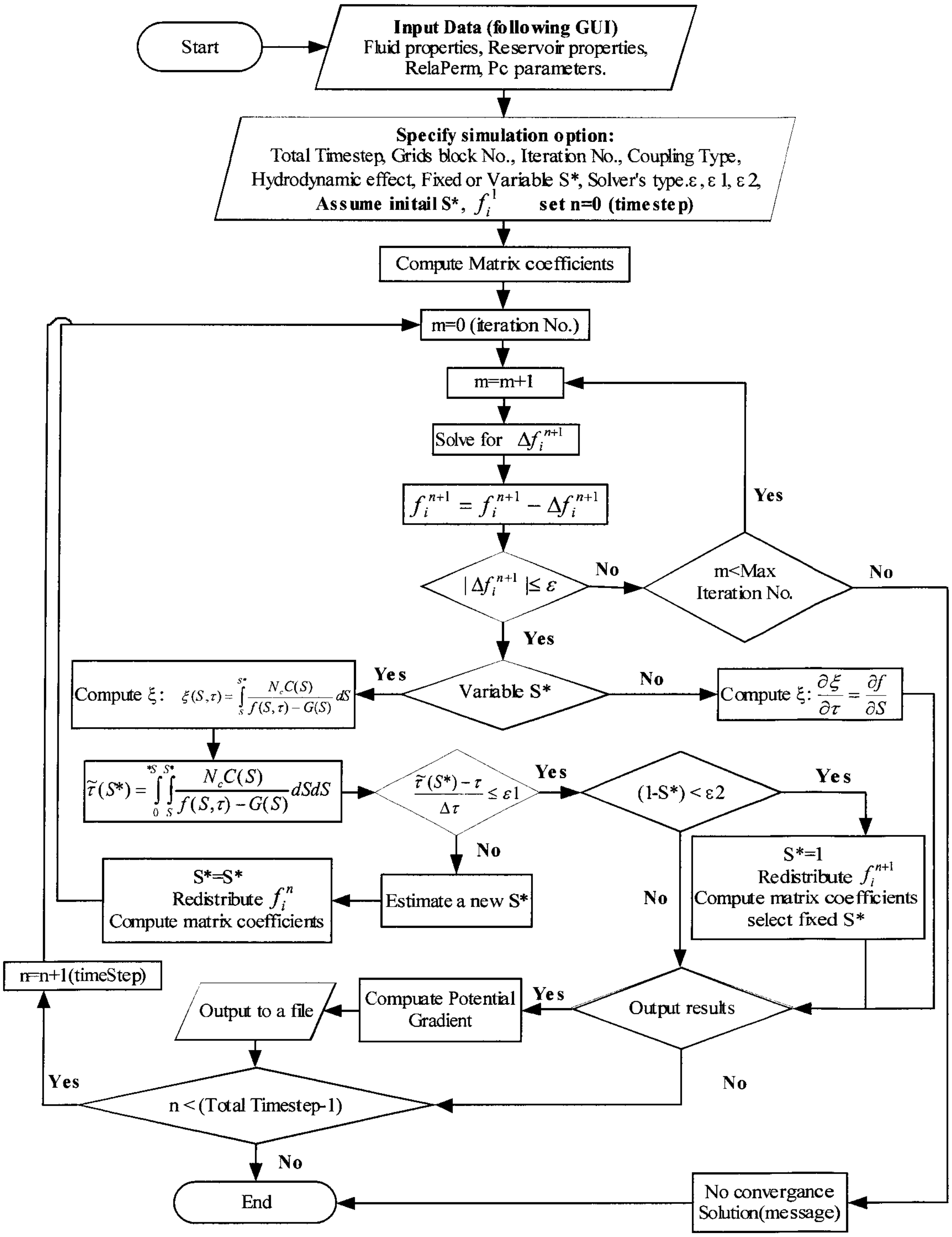
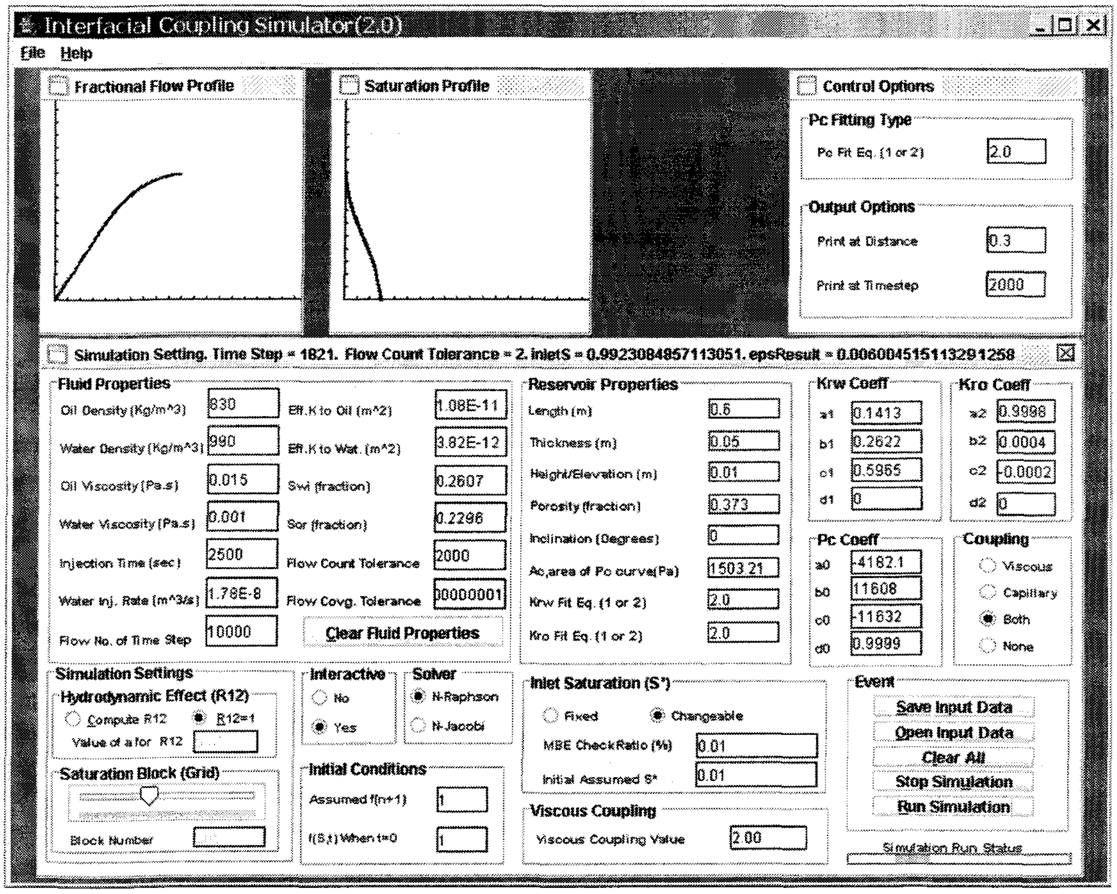

Start
Input Data (following GUI)
Fluid properties, Reservoir properties,
RelaPerm, Pc parameters.
Specify simulation option:
Total Timestep, Grids block No., Iteration No., Coupling Type,
Hydrodynamic effect, Fixed or Variable S*, Solver's type.ε, ε 1, ε 2,
Assume initail S*, f_i^1 set n=0 (timestep)
Compute Matrix coefficients
m=0 (iteration No.)
m=m+1
Solve for Δf_i^{n+1}
$f_i^{n+1} = f_i^{n+1} - \Delta f_i^{n+1}$
$|\Delta f_i^{n+1}| \leq \varepsilon$
No
Yes
m<Max Iteration No.
Yes
No
Variable S*
Yes
No
Compute ξ: $\xi(S,\tau) = \int_S^{S^*} \frac{N_c C(S)}{f(S,\tau) - G(S)} dS$
Compute ξ: $\frac{\partial \xi}{\partial \tau} = \frac{\partial f}{\partial S}$
$\tilde{\tau}(S^*) = \int_0^{S^*}\int_S^{S^*} \frac{N_c C(S)}{f(S,\tau) - G(S)} dS dS$
$\frac{\tilde{\tau}(S^*) - \tau}{\Delta \tau} \leq \varepsilon 1$
Yes
(1-S*) < ε2
Yes
No
S*=S*
Redistribute f_i^n
Compute matrix coefficients
Estimate a new S*
No
S*=1
Redistribute f_i^{n+1}
Compute matrix coefficients
select fixed S*
n=n+1(timeStep)
Output to a file
Compute Potential Gradient
Yes
Output results
No
Yes
n < (Total Timestep-1)
No
End
No convergance Solution(message)

Figure 5-1: The flow chart for the Interfacial Coupling Simulator (2.0)

For the reader's convenience, the source codes of the Interfacial Coupling Simulator (2.0) are given in Appendix D. Moreover, the source codes (.java files) and their byte code representations (.class files), the packaged executable JAR (Java Archive) file and an example input data file are burned into the CD-ROM attached at the back of this thesis.

5.6.1 Data input and simulation control options

Similar to Interfacial Coupling Simulator (1.0), a modified graphical user interface (GUI), as shown in Figure 5-2, is employed to facilitate data input and simulation control options.



Pc Fitting Type

Pc Fit Eq. (1 or 2): 2.0

Output Options

Print at Distance: 0.3

Print at Timestep: 2000

Simulation Setting. Time Step = 1821. Flow Count Tolerance = 2. inletS = 0.9923084857113051. epsResult = 0.006004515113291258

Fluid Properties

Oil Density (Kg/m^3): 830

Water Density (Kg/m^3): 990

Oil Viscosity (Pa.s): 0.015

Water Viscosity (Pa.s): 0.001

Injection Time (sec): 2500

Water Inj. Rate (m^3/s): 1.78E-8

Flow No. of Time Step: 10000

Eff.K to Oil (m^2): 1.08E-11

Eff.K to Wat. (m^2): 3.82E-12

Swi (fraction): 0.2607

Sor (fraction): 0.2296

Flow Count Tolerance: 2000

Flow Covg. Tolerance: 00000001

Reservoir Properties

Length (m): 0.6

Thickness (m): 0.05

Height/Elevation (m): 0.01

Porosity (fraction): 0.373

Inclination (Degrees): 0

Ac,area of Pc curve(Pa): 1503.21

Krw Fit Eq. (1 or 2): 2.0

Kro Fit Eq. (1 or 2): 2.0

Krw Coeff: a1 0.1413, b1 0.2622, c1 0.5965, d1 0

Kro Coeff: a2 0.9998, b2 0.0004, c2 -0.0002, d2 0

Pc Coeff: a0 -4182.1, b0 11608, c0 -11632, d0 0.9999

Coupling: ○ Viscous ○ Capillary ● Both ○ None

Simulation Settings

Hydrodynamic Effect (R12): ○ Compute R12 ● R12=1

Value of a for R12

Saturation Block (Grid)

Block Number

Interactive: ○ No ● Yes

Solver: ● N-Raphson ○ N-Jacobi

Initial Conditions

Assumed f(n+1): 1

f(S,t) When t=0: 1

Inlet Saturation (S*): ○ Fixed ● Changeable

MBE CheckRatio (%): 0.01

Initial Assumed S*: 0.01

Viscous Coupling

Viscous Coupling Value: 2.00

Figure 5-2: Graphical user interface of the Interfacial Coupling Simulator (2.0)

As can be seen from Figure 5-2, the basic reservoir (or core) and fluid properties can be input directly into the graphical user interface. The relative permeability data and capillary pressure data are input in the form of the coefficients of the corresponding data fitting equations. Data fitting equations accepted by the simulator are provided in the following two sections.

5.6.1.1 Relative permeability fitting equations

In the simulator, the relative permeability is defined by using the effective permeability at the end point. Therefore, the fitted relative permeability curves must meet the requirement that the wetting-phase relative permeability has to vary between zero and one, while the nonwetting-phase relative permeability has to vary between one and zero when the normalized wetting phase is equal to zero and one, respectively. Satisfying the above requirements, two types of fitting equations are provided in the simulator.

The first type fitting equations were proposed by Bentsen (1976), and they have the following forms:

$$k_{r1} = \left(\frac{a_1 + b_1 \times (1-S)}{a_1 + (1-S)} \right) \times S^{c_1}, \tag{5.65}$$

and

$$k_{r2} = \left(\frac{a_2 + b_2 \times S}{a_2 + S} \right) \times (1-S)^{c_2}. \tag{5.66}$$

The second type fitting equations are polynomial equations with a certain constraint on the coefficients to ensure that the requirement stated above can be met. These equations and coefficients are:

$$k_{r1} = a_1 S + b_1 S^2 + c_1 S^3 + d_1 S^4 \tag{5.67}$$

and

$$k_{r2} = a_2(1-S) + b_2(1-S)^2 + c_2(1-S)^3 + d_2(1-S)^4 \;. \tag{5.68}$$

where $a_1 + b_1 + c_1 + d_1 = 1$; $a_2 + b_2 + c_2 + d_2 = 1$

The first and the second type of fitting equations are symbolized by 1 and 2, respectively. Both fitting equation type and fitting coefficients are needed when inputting the relative permeability data into the simulator.

5.6.1.2 Capillary pressure fitting equation

Similarly, the simulator provides two types of fitting equations for capillary pressure data. The first type fitting equation, denoted by 1 in the simulator, was proposed by Bentsen (1976). It has the following form:

$$P_c = \frac{a_0(1-S) + b_0(1-S^2)}{1 + c_0 S + d_0 S^2} \,. \tag{5.69}$$

The second type fitting equation, symbolized by 2 in the simulator, is a third order polynomial equation. This equation is written in the following form in correspondence to the symbols specified in the GUI of the simulator.

$$P_c = a_0 S^3 + b_0 S^2 + c_0 S + d_0 \,. \tag{5.70}$$

5.6.1.3 Simulation control options

Most of the simulation control options shown on the GUI are specified clearly and are easy to understand. More explanations are needed for the following options:

1. Inlet Saturation (S*): in the Inlet Saturation (S*) option section of the GUI, one needs to select the "Changeable" option and specify the value for both the initial assumed inlet saturation and material balance check ratio. The material balance check ratio is defined as $\left| \frac{\tilde{\tau}(S^*) - \tau}{\Delta \tau} \right|$. The numerator in the definition is the

absolute value of the difference between the calculated injected normalized pore volume and the actual injected normalized pore volume, while the denominator is the injected normalized pore volume of one time step. At every time step, the simulator will calculate the actual value of $|\frac{\tilde{\tau}(S^*)-\tau}{\Delta\tau}|$ and compare it with the preset value, ε_1, in the GUI. If the condition dictated by Equation (5.71) is satisfied, the simulation will proceed to the next time step; otherwise, the simulator will adjust the value of inlet saturation, S^*, until Equation (5.71) is met.

$$|\frac{\tilde{\tau}(S^*)-\tau}{\Delta\tau}|\le\varepsilon_1 . \qquad (5.71)$$

Once the inlet saturation, S^*, reaches its maximum value, 1, the "Fixed" option will be selected automatically. From this point on, the saturation profile is calculated based on the frontal advance equation rather than the fractional flow equation that is used when the "Changeable" option is selected.

2. Viscous Coupling: as to the "Viscous Coupling" option, it needs to be specified that the value input here is not the coefficient, c, in Equations (5.16) and (5.17). Rather, the product of the input value and the square of the porosity of the core, ϕ^2, represents the coefficient, c.

3. Flow Count Tolerance and Flow Convergence Tolerance: Flow Count Tolerance denotes the maximum iteration steps and Flow Convergence Tolerance stands for the maximum difference between consecutive iterations. Generally, in order to make sure that sufficient iterations are performed, the value for Flow Count Tolerance should be set as a big number, especially when the Newton-Jacobi solver is selected to conduct simulations. On the contrary, the value for the Flow Convergence Tolerance should be set as a small number to meet the accuracy requirement.

4. Output options: simulation results for the last time step are saved automatically in the output data file as set by the simulator. In addition, the simulator provides two

more output options: Print at Distance and Print at Time step (as shown in Figure 5-2). Using these two options, one can require the simulator to save the simulation results that correspond to a certain normalized travel distance of the flood front, or that correspond to a certain time step.

5. Interactive: if the "yes" button in the "interactive" panel of the GUI of Figure 5-2 is enabled, the fractional flow and the saturation profiles are plotted and can be viewed dynamically as the computation progresses. Otherwise, if the "no" button is selected, the fractional flow and the saturation profiles cannot be viewed as computation progresses.

5.6.2 Description of source codes

During the development of the Interfacial Coupling Simulator (2.0), the approach used in Ayodele's Interfacial Coupling Simulator [Ayodele (2004)] to organize the java classes was employed. However, the contents of the classes were modified, somewhat. Especially, the main computational class of the simulator, IcsClass.java, was improved significantly, which can be found in Appendix D. The description of the functions of the various classes is given below.

1. Ics.java: This class is the main class that initiates the Interfacial Coupling Simulator.

2. IcsFrame.java: This class is used to construct the background GUI that holds the other GUIs.

3. IcsClass.java: This class has three main functions which are: a), forming sub-GUIs in the main GUI constructed by IcsFrame.java, b), receiving input data from the GUI or from the input data file and performing computations, c), outputting simulation results.

4. FlowView.java: This class is used to display the dynamic view of the fractional flow and saturation profiles as the computation progresses.

5. IcsFrame_AboutBox.java: This class contains codes for showing information about the author(s), company /university and software version in a dialog box.

5.7 Validation of the simulator

The validations in this section are focused mainly on two issues: (a) whether or not the "Newton-Raphson Solver" and the "Newton-Jacobi solver" give the same simulation results as predicted on the basis of mathematical analysis in Section 5.5.1, (b) whether or not the simulator can work properly under the Variable Inlet Saturation Conditions. These two issues are validated by using Ayub's experimental data (Data Set 2) [Ayub (2000)].

5.7.1 Experimental Data

Data set 2 of Ayub's experimental data [Ayub (2000)] is shown in Tables 5-1 and 5-2. The experimental data were processed for simulation purposes and they are shown in Table 5-3. Figure 5-3 gives the measured USCO saturation profile and its fitted curve. Figure 5-4 shows the measured capillary pressure curve which was fitted by Equation (5.69). Figure 5-5 represents the relative permeability curves that were fitted using polynomial models described by Equations (5.67) and (5.68) by utilizing the least squares method with a constraint on the coefficients. The fitted coefficients are listed in Table 5-4.

Core measurement length	0.6 m
Core width	0.05 m
Core height	0.01 m
Core inclination	0 Degree
Sand type	Ottawa silicate sand (80-120) mesh)
Average porosity	0.373
Core pore volume	0.0001119 m^3
Wetting phase	Distilled Water
Non-wetting phase	Kerosene+Light Mineral Oil
Wetting phase density at room temperature	990 Kg/m^3
Wetting phase viscosity at room temperature	0.001 Pa.s
Non-wetting phase density at room temperature	830 Kg/m^3
Non-wetting phase viscosity at room temperature	0.015 Pa.s
Flow type	SSCO, USCO
Average total displacement rate (USCO) , q	1.78E-8 m^3/s
Initial wetting phase saturation S_{1i}	0.2607
Residual non-wetting phase saturation S_{2r}	0.2296
Cocurrent effective perm. to non-wetting phase at S_{1i}	10.8E-12 m^2
Cocurrent effective perm. to wetting phase at S_{2r}	3.82E-12 m^2

Table 5-1: Properties of Core and Fluids [Ayub (2000)]

Wetting phase saturation	Mobility Data (m/Pa.S)		Capillary pressure (Pa)	Normalized distance
	Non-wetting Phase	Wetting Phase		
0.7704	0.0000E+00	6.3650E-09	299.8576	0
0.7537	3.9263E-11	5.8673E-09	322.804	0.05
0.7381	7.6182E-11	5.4243E-09	345.6092	0.1
0.7226	1.1264E-10	5.0101E-09	369.667	0.15
0.7069	1.4963E-10	4.6125E-09	396.0365	0.2
0.6909	1.8748E-10	4.2287E-09	425.4926	0.25
0.6746	2.2599E-10	3.8613E-09	458.5382	0.3
0.6582	2.6470E-10	3.5148E-09	495.398	0.35
0.6419	3.0300E-10	3.1937E-09	536.0353	0.4
0.6261	3.4038E-10	2.9002E-09	580.2505	0.45
0.6107	3.7660E-10	2.6341E-09	627.9246	0.5
0.5958	4.1188E-10	2.3915E-09	679.4603	0.55
0.5808	4.4710E-10	2.1650E-09	736.457	0.6
0.5652	4.8400E-10	1.9439E-09	802.6961	0.65
0.5477	5.2535E-10	1.7150E-09	885.6519	0.7
0.5265	5.7517E-10	1.4643E-09	999.1273	0.751
0.4995	6.3888E-10	1.1813E-09	1168.4592	0.801
0.4636	7.2355E-10	8.6460E-10	1441.6735	0.85
0.4151	8.3804E-10	5.3053E-10	1914.4772	0.9
0.3493	9.9322E-10	2.1671E-10	2787.8117	0.95
0.2607	1.2022E-09	0.0000E+00	4502.9064	1

Table 5-2: Data set 2 of Ayub's [Ayub (2000)] experiment

Normalized Saturation	K_{r2}	K_{r1}	P_c(pa)	Normalized Distance
1	0	1	299.8576	0
0.967235629	0.032659	0.921807	322.804	0.05
0.93662939	0.063369	0.852207	345.6092	0.1
0.906219345	0.093695	0.787133	369.667	0.15
0.875416912	0.124463	0.724666	396.0365	0.2
0.844025898	0.155947	0.664368	425.4926	0.25
0.812046302	0.18798	0.606646	458.5382	0.3
0.779870512	0.22018	0.552207	495.398	0.35
0.747890916	0.252038	0.50176	536.0353	0.4
0.71689229	0.283131	0.455648	580.2505	0.45
0.686678438	0.313259	0.413841	627.9246	0.5
0.657445556	0.342605	0.375727	679.4603	0.55
0.62801648	0.371902	0.340141	736.457	0.6
0.597410241	0.402595	0.305405	802.6961	0.65
0.563076319	0.436991	0.269442	885.6519	0.7
0.521483225	0.478431	0.230055	999.1273	0.751
0.468510889	0.531426	0.185593	1168.459	0.801
0.3980773	0.601855	0.135837	1441.674	0.85
0.302923288	0.697089	0.083351	1914.477	0.9
0.173827742	0.826169	0.034047	2787.812	0.95
0	1	0	4502.906	1

Table 5-3: Data processed from Table 5-2

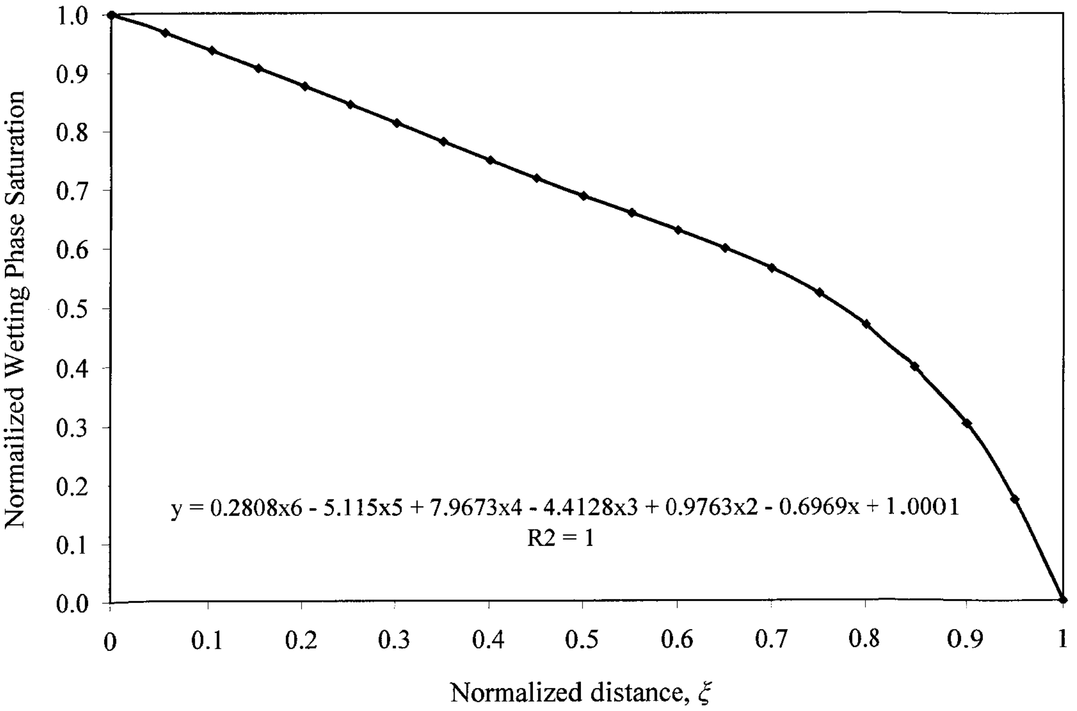


Figure 5-3: Fitted USCO saturation-distance profile

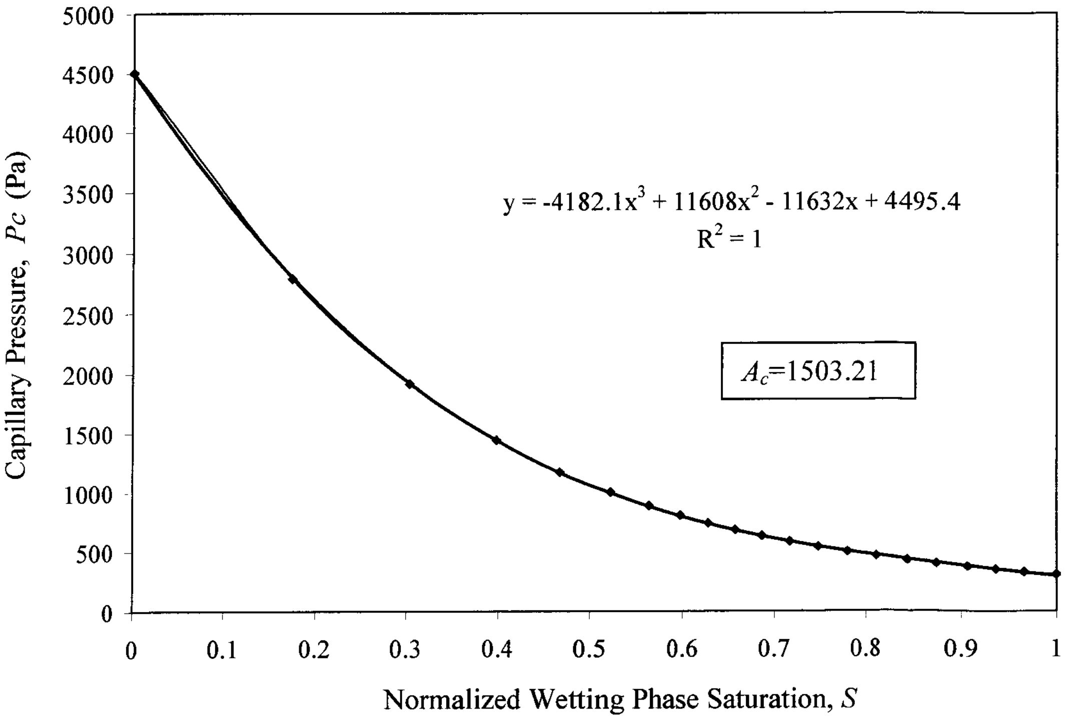


Figure 5-4: Fitted capillary pressure curve

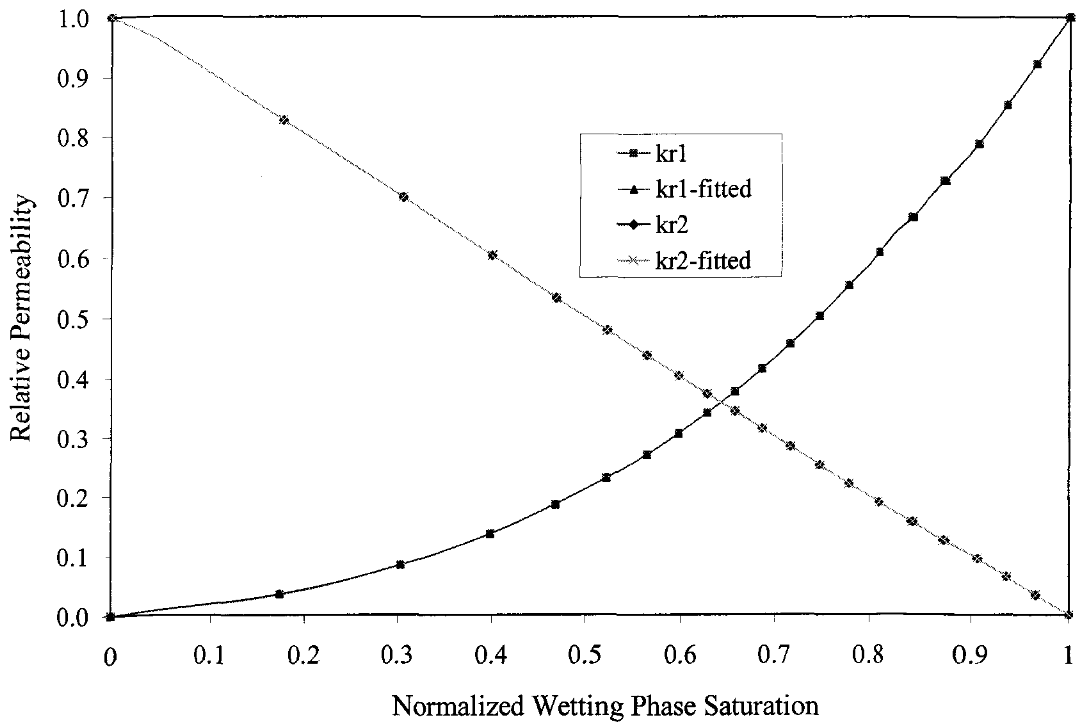


Figure 5-5: Fitted relative permeability curves

Wetting phase		Non-wetting phase	
a_l	0.1413	a_2	0.9998
b_l	0.2622	b_2	0.0004
c_l	0.5965	c_2	-0.0002
d_l	0	d_2	0
Sum(Error^2)	3.9033E-06	Sum(Error^2)	6.6665E-08

Table 5-4: Fitted coefficients for k_{r1} and k_{r2}

5.7.2 Comparison of Newton-Raphson and Newton-Jacobi solvers

In order to check whether the “Newton-Raphson Solver” and the “Newton-Jacobi solver” give the same results, on the basis of the data described in Section 5.7.1, simulation runs, with the settings specified in Table 5-5, were carried out by using these two solvers. The simulation results and the difference between the simulation results from the two solvers are plotted in Figure 5-6.

ΔS	S*/40	$\Delta \tau$	0.000156
Flow count tolerance	20000	Flow convergence tolerance	1E-10
Coupling effects	No	Interactive	No
Initial assumed S*	0.01	MBE Check Ratio	0.1%

Table 5-5: Simulation settings

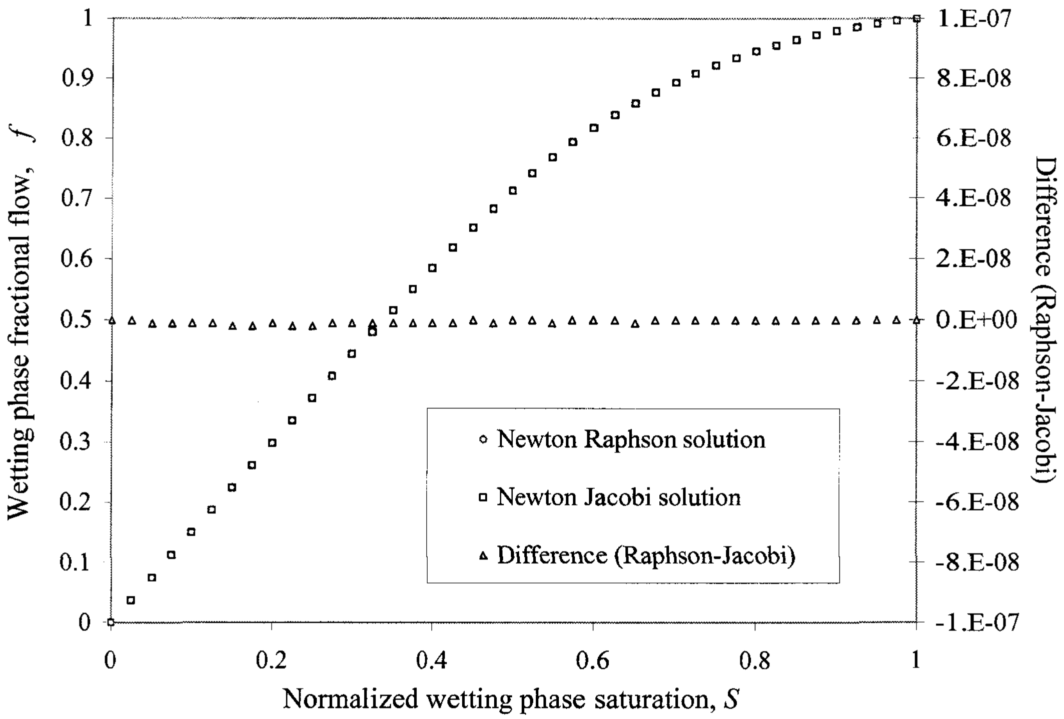


Figure 5-6: Comparison of Newton-Raphson and Newton-Jacobi solvers

As can be seen from Figure 5-6, the simulation results of these two solvers are essentially the same. However, the computation time of the Newton-Jacobi solver, 9 seconds, is much longer than that of the Newton-Raphson solver, 3 seconds. Moreover, the difference between the computation time gets bigger as the saturation grid size, ΔS, becomes smaller. Therefore, it is recommended to use the Newton-Raphson solver to conduct simulations.

5.7.3 Validation of the simulator under Variable Inlet Saturation Conditions

As mentioned in Section 5.4, the correct solution of Bentsen's equation has to satisfy both the fractional flow equation and the frontal advance equation. Because the fractional flow equation was used to calculate the saturation profile when the Variable Inlet Saturation Conditions were applied, one has to use the frontal advance equation to check the validity of the solution. If the LHS of the frontal advance equation [Equation (5.27)], $\partial\xi/\partial\tau$, is equal to the RHS of it, $\partial f/\partial S$, when $f(S,\tau)$, the solution to Bentsen's equation, and $\xi(S,\tau)$, the solution of the fractional flow equation [Equation (5.26)], are used, it can be concluded that the simulator works well under the Variable Inlet Saturation Conditions and the simulation results are correct.

Figure 5-7 is a validation example obtained by using the validation method discussed above. The data used were taken from the simulation results by using Ayub's experimental data [Ayub (2000)] with the Variable Inlet Saturation Conditions as the initial and boundary conditions. As can be seen from this figure, the match is good with only a very small difference between $\partial\xi/\partial\tau$ and $\partial f/\partial S$ at some points, which may have resulted from using numerical derivatives and which may be removed by properly selecting $\Delta\tau$ and ΔS.

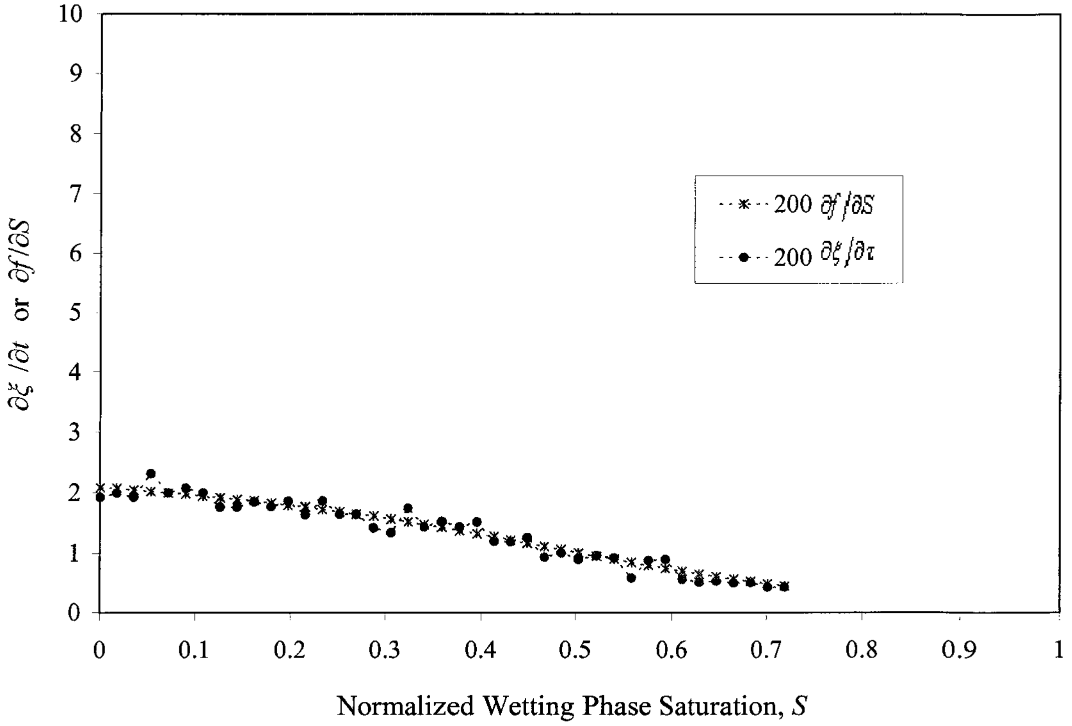


Figure 5-7: Validation of Bentsen's equation solution with Variable Inlet Saturation Conditions at 200 seconds

The material balance error value, which is expressed as $|\frac{\tilde{\tau}(S^*)-\tau}{\tau}|$, is very small. The results obtained at four different arbitrarily chosen injection times are shown in Table 5-6.

Material balance error	50 Seconds	100 Seconds	200 Seconds	400 Seconds
$|\frac{\tilde{\tau}(S^*)-\tau}{\tau}|$	0.01%	0.00%	0.00%	0.00%

Table 5-6: Material balance error at different injection times

Moreover, this error value can be reduced by setting a more strict material balance check criterion [making ε_1 smaller in Equation (5.71)], which will result in a longer time to run the simulation.

The propagation of the saturation profile with time is shown in Figure 5-8. It is clear that S^* is increasing with time, which is consistent with experimental observations, as can be seen in Figure 5-9. Consequently, it appears that the simulator is able to solve Bentsen's equation correctly under the Variable Inlet Saturation Conditions.

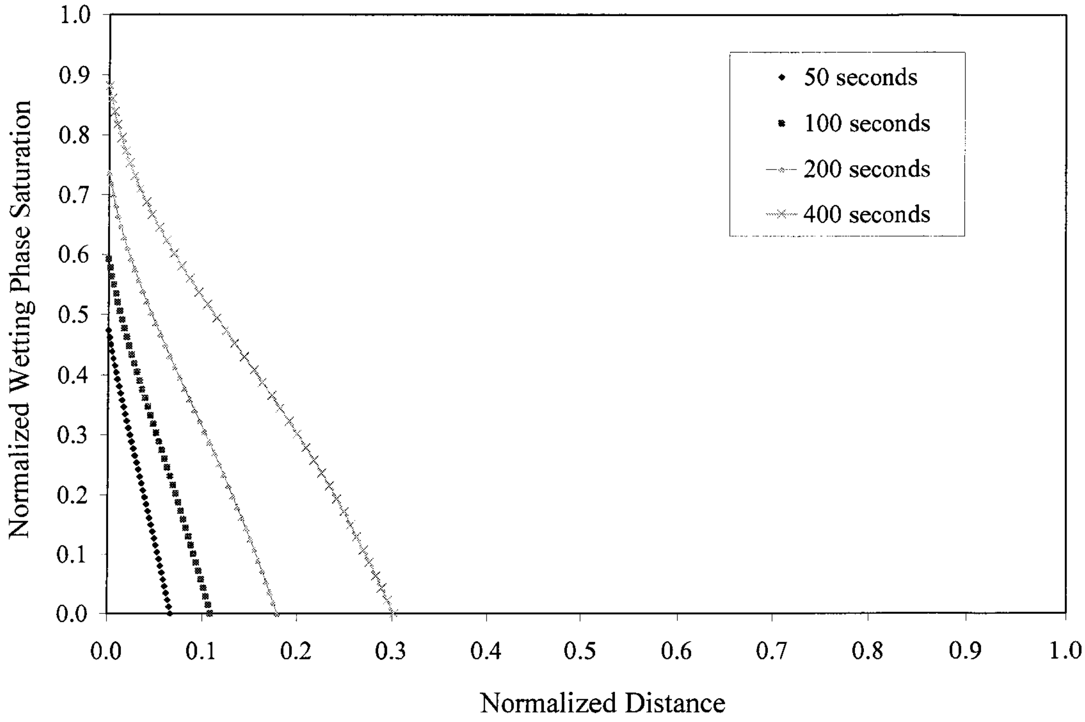


Figure 5-8: Saturation profiles for Variable Inlet Saturation Conditions

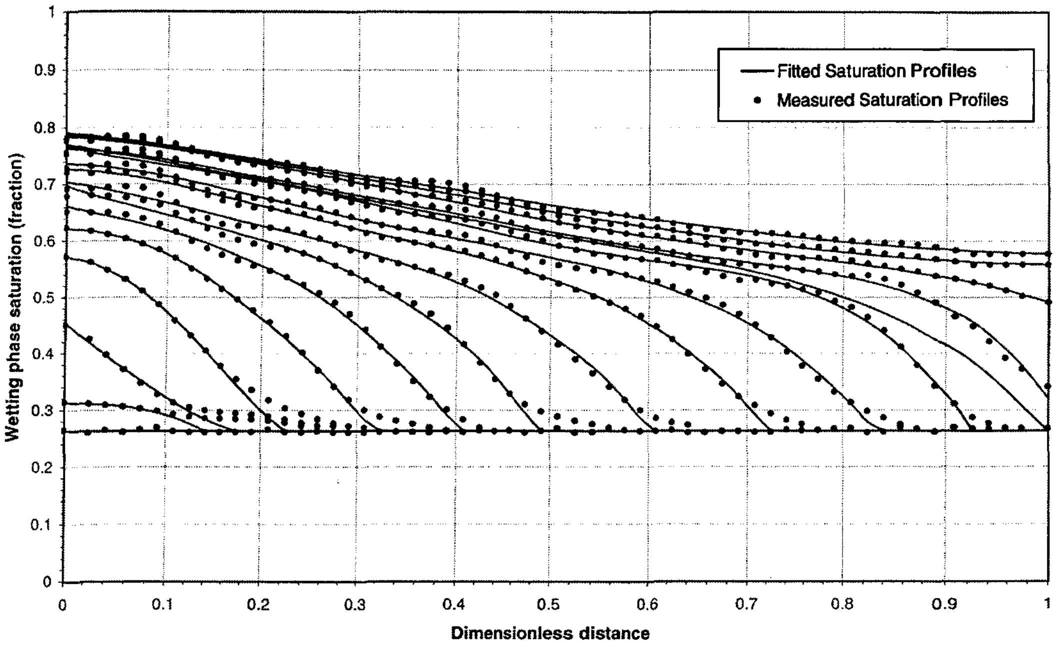


Figure 5-9: Saturation profiles obtained during USCO [After Ayub (2000)]

CHAPTER 6

Analysis of experimental and simulation results

6.1 Introduction

In this chapter, history match analyses were performed by utilizing the simulator developed in Chapter 5, the experimental data from both Ayub's thesis [Ayub (2000)] and the laboratory experiments conducted in Chapter 4. The history matches were performed on saturation and pressure profiles, but not on fractional flow profiles. The reason for this is that a fractional flow profile can not be measured directly from experiments; instead, it is obtained from saturation profiles [the calculation procedures can be found in the paper by Sarma and Bentsen (1989)]. If good matches can be achieved on the saturation profiles, the corresponding fractional flow profiles should have good matches as well. In order to minimize the error introduced through the numerical solving process, acceptable values for saturation grid size, ΔS, and time step size, $\Delta \tau$, were found through preliminary simulation runs. In addition, sensitivity analyses were carried out and several conclusions were drawn as well.

6.2 Experimental data and related processing

Three sets of experimental data were used to carry out the history match analyses. For the sake of convenience, these experimental data sets are referred to as Data Groups A, B and C which are: (A) Data Set 2 of Ayub's experimental data [Ayub (2000)], (B) Scan 27 of USCO flow experiment Run 6, and (C) Scan 12 of USCO flow experiment Run 7, respectively. Data Group A is presented in Tables 5-1 through 5-4, and Figures 5-3 through 5-5. Detailed descriptions can be found in Section 5.6.1. As discussed in Chapter 4, USCO flow experiments Run 6 and 7 were conducted on the same sand fluid system and have the same basic core-fluid data. These basic core-fluid data are presented in Table 4-1, Figure 4-12 and Figure 4-13. The saturation profiles of USCO flow experiments Run 6 and 7 are presented in Figures 4-16 and 4-17, respectively. The pressure profiles of Data Group B and C are shown in Figures 4-18 and 4-19, respectively.

In order to obtain input data for the simulator, the relative permeability curves in Figure 4-13 were fitted using the polynomial models described by Equations (5.67) and (5.68). The least squares method, with a constraint on the coefficients, was utilized. The fitted relative permeability curves are shown in Figure 6-1 and the fitted coefficients are listed in Table 6-1.

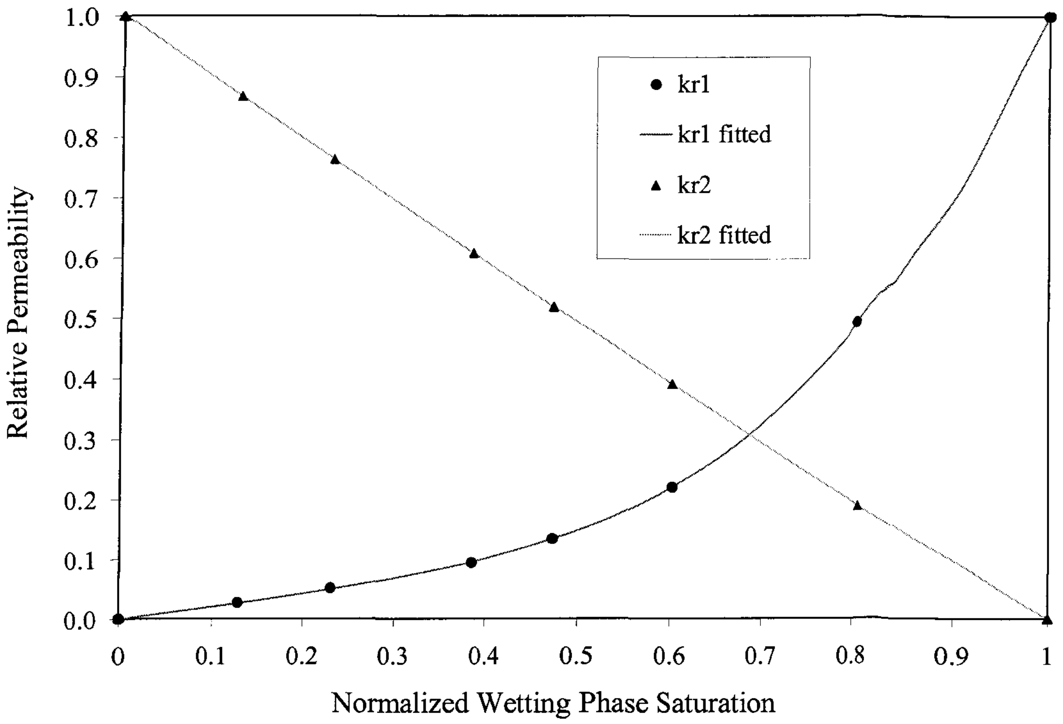


Figure 6-1: Fitted relative permeability curves

Wetting phase		Non-wetting phase	
a_1	0.1871	a_2	0.9765
b_1	0.1643	b_2	0.0110
c_1	-0.4314	c_2	0.0125
d_1	1.08	d_2	0
$a_1 + b_1 + c_1 + d_1$	1.0000	$a_2 + b_2 + c_2 + d_2$	1.0000
Sum(Error^2)	3.69944E-06	Sum(Error^2)	1.37E-07

Table 6-1: Fitted coefficients for k_{r1} and k_{r2}

6.3 Selection of saturation grid size and time step size

When solving numerically a partial differential equation using the finite difference approximation technique, two types of errors, round-off error and truncation error, can be introduced. Moreover, there is always a tradeoff between rounding error and truncation error. That is, when the mesh sizes, which are the saturation grid size and the time step size in this study, are reduced, the truncation errors are reduced also, whereas the rounding error becomes larger, and vice versa. Practically, it is hard to get an optimal saturation grid size and time step size. Therefore, it was decided to select an acceptable saturation grid size and time step size which can give acceptable results.

In this study, the selection of an acceptable saturation grid size, ΔS, or time step size, $\Delta\tau$, was achieved by observing the computed results (saturation profiles) when varying ΔS or $\Delta\tau$. When saturation profiles converge sufficiently, an acceptable ΔS or $\Delta\tau$ can be determined. The reason for using saturation profiles to monitor the influence of ΔS and $\Delta\tau$ is that saturation profiles are more sensitive than fractional flow profiles; that is, a small change on fractional flow profiles results in a large change on the corresponding saturation profiles.

It should be noted that an acceptable saturation grid size, ΔS, and time step size, $\Delta\tau$, can be different when different sand-fluid systems are simulated using a given simulator. Therefore, acceptable values of ΔS and $\Delta\tau$ were determined, respectively, for the two sand-fluid systems used in this chapter. For Data Group A, the saturation profiles for different ΔS at a given $\Delta\tau$ are shown in Figure 6-2, while Figure 6-3 presents the saturation profiles for different $\Delta\tau$ at a given ΔS. For Data Groups B and C, the saturation profiles both for different ΔS at a given $\Delta\tau$ and for different $\Delta\tau$ at a given ΔS are shown in Figures 6-4 and 6-5, respectively. As can seen from Figures 6-2 through 6-5, the saturation grid size and time step size shown do not make too much difference on the saturation profiles. That is to say, these saturation grid sizes and time step sizes are all acceptable. In fact, because the double precision

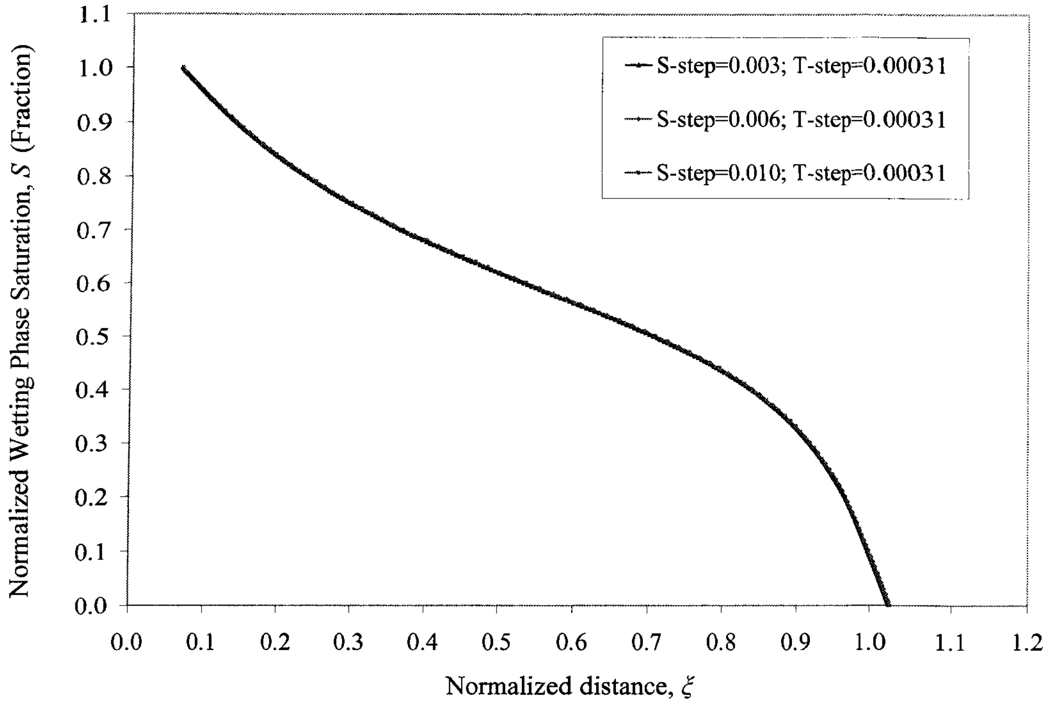


Figure 6-2: Saturation profiles for various ΔS at a given $\Delta\tau$ [for Data Group A]

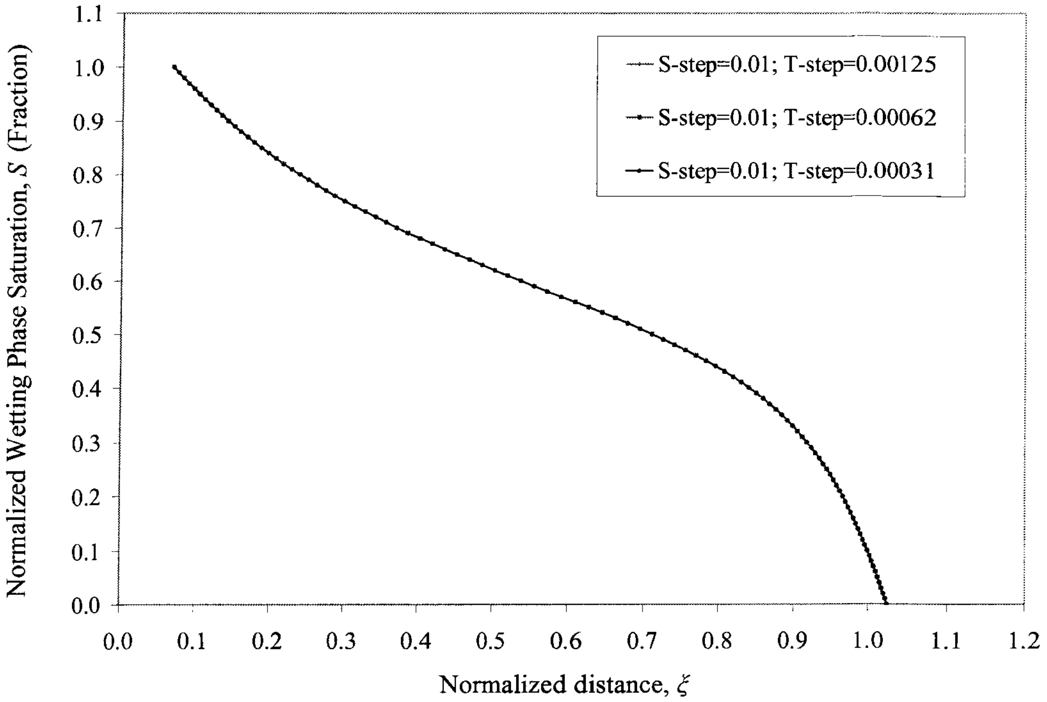


Figure 6-3: Saturation profiles for various $\Delta\tau$ at a given ΔS [for Data Group A]

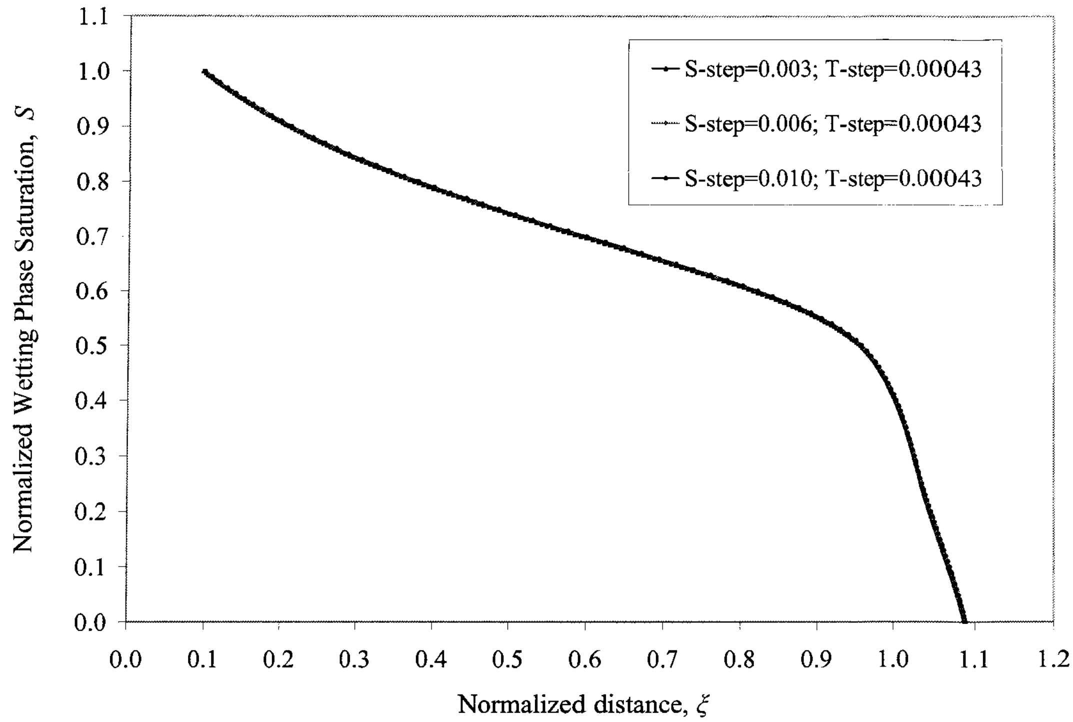


Figure 6-4: Saturation profiles for various ΔS at a given $\Delta\tau$ [for Data Groups B and C]

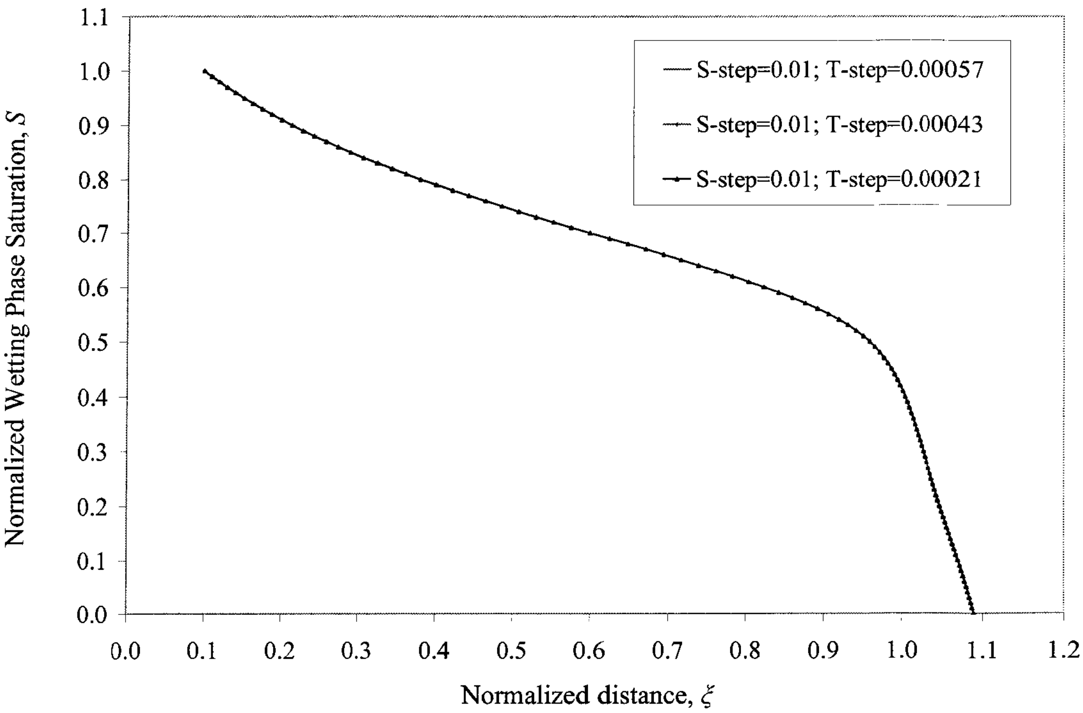


Figure 6-5: Saturation profiles for various $\Delta\tau$ at a given ΔS [for Data Groups B and C]

data type, which can use data absolute values as small as 4.9E-324, was used in the simulator, one may note that the round-off error should be quite small when considering the type of equations to be solved. Therefore, saturation grid sizes and time step sizes which are a couple of orders of magnitude less than the values shown in Figures 6-4 and 6-5 should be acceptable also.

6.4 History matches

In this section, history matches were performed on saturation profiles and pressure profiles by utilizing Data Groups A, B and C. With the corresponding input data described in Section 6.2, and the acceptable values of ΔS and $\Delta \tau$ provided in Table 6-2, several simulation runs, with and without both capillary and viscous coupling effects, were carried out to seek matches between experimental and simulation results. When performing simulations, if the interfacial coupling effects, either viscous, capillary or both viscous and capillary coupling effects, are considered, the simulation results are based on the Modified Transport Equations model; if no interfacial coupling effects are taken into account, the simulation results are based on the traditional transport equations model. In the simulation runs with both capillary and viscous coupling effects considered, the hydrodynamic effect was neglected and the viscous coupling value was set as 2, which is a theoretical value found by Ayodele (2004).

Step size	Data Group A	Data Group B	Data Group C
ΔS	0.01	0.01	0.01
$\Delta \tau$	0.00031	0.00027	0.00021

Table 6-2: Acceptable values of step size used in the history match simulation

6.4.1 History matches based on the Modified Transport Equations

Based on the Modified Transport Equations model, that is, with the ‘Both coupling’ option in the GUI (see Figure 5-2) selected, several simulation runs were conducted and matches between experimental and simulation results were obtained. The matches of the saturation and pressure profiles for Data Groups A, B and C are shown in Figures 6-6 through 6-11. The time values shown in the legend of the figures for the saturation profile match denote the injection time corresponding to the saturation profiles. The injection time for the experimental saturation profiles was either recorded directly during the experiment, as was the case for Data Group B or C, or calculated by using the injection rate and the area under the measured saturation profile, as was the case for Data Group A. The injection time for the simulation saturation profiles was taken from the simulation results.

As can be seen from Figures 6-6, 6-8 and 6-10, the discrepancy between the injection time for the experimental saturation profiles and that for the simulation saturation profiles is quite small, and the shapes of the experimental and simulation saturation profiles are fairly similar. As for the matches of the pressure profiles in Figures 6-7, 6-9 and 6-11, they are not too bad, but not as good as they should be.

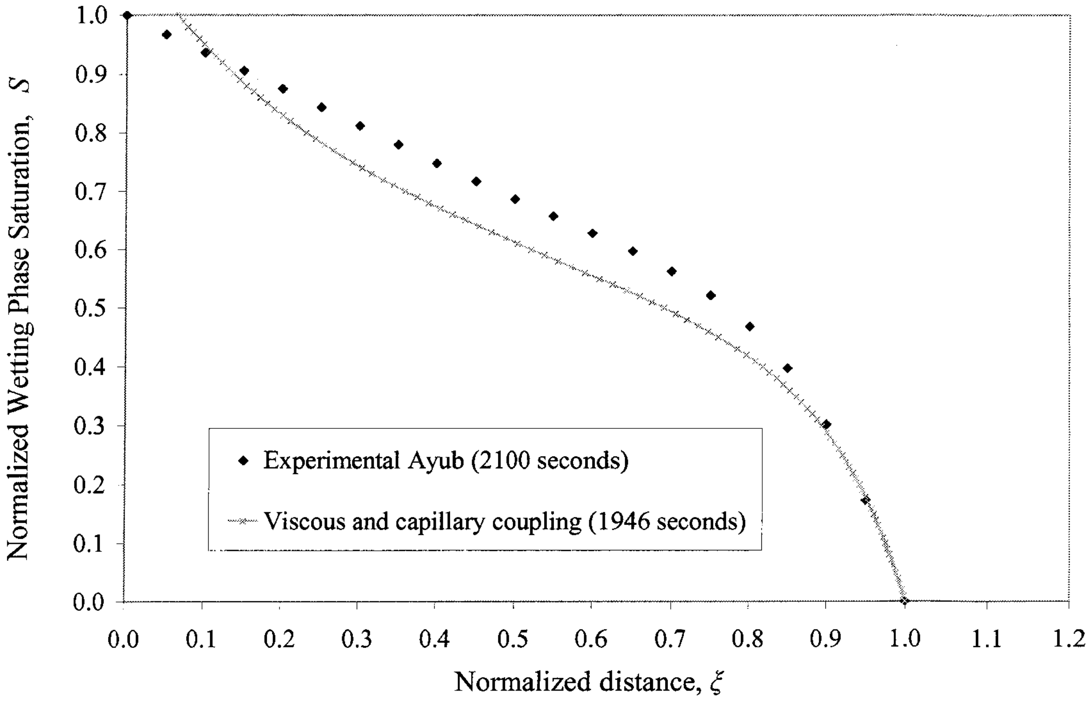


Figure 6-6: Match of saturation profiles when both viscous and capillary coupling effects are considered [Data Group A]

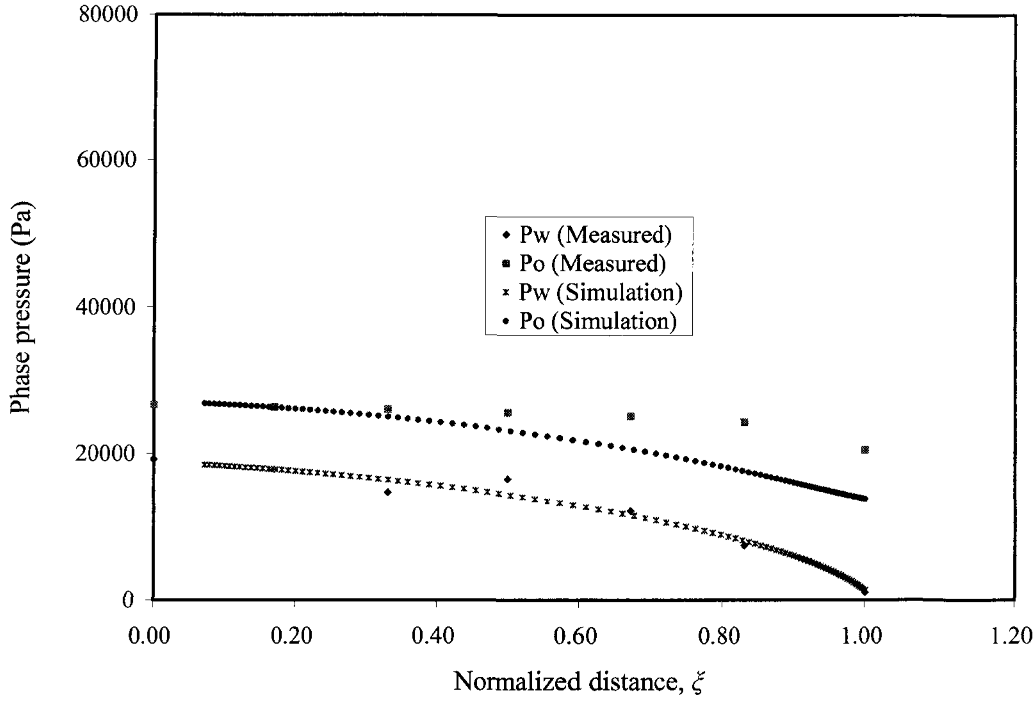


Figure 6-7: Match of pressure profiles when both viscous and capillary coupling effects are considered [Data Group A]

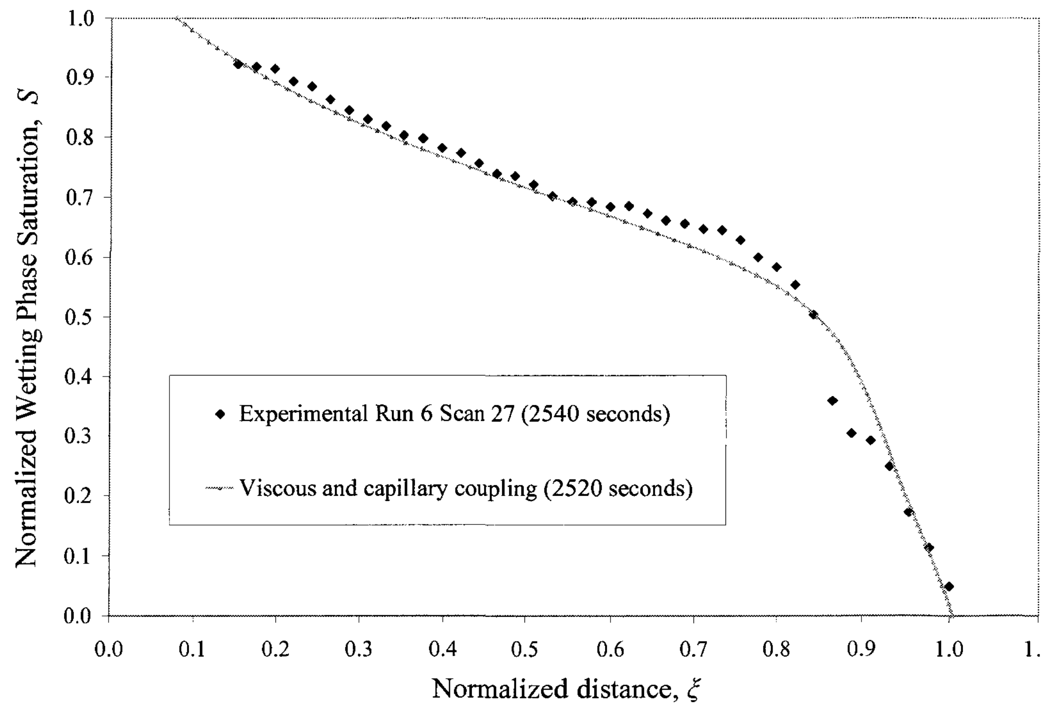


Figure 6-8: Match of saturation profiles when both viscous and capillary coupling effects are considered [Data Group B]

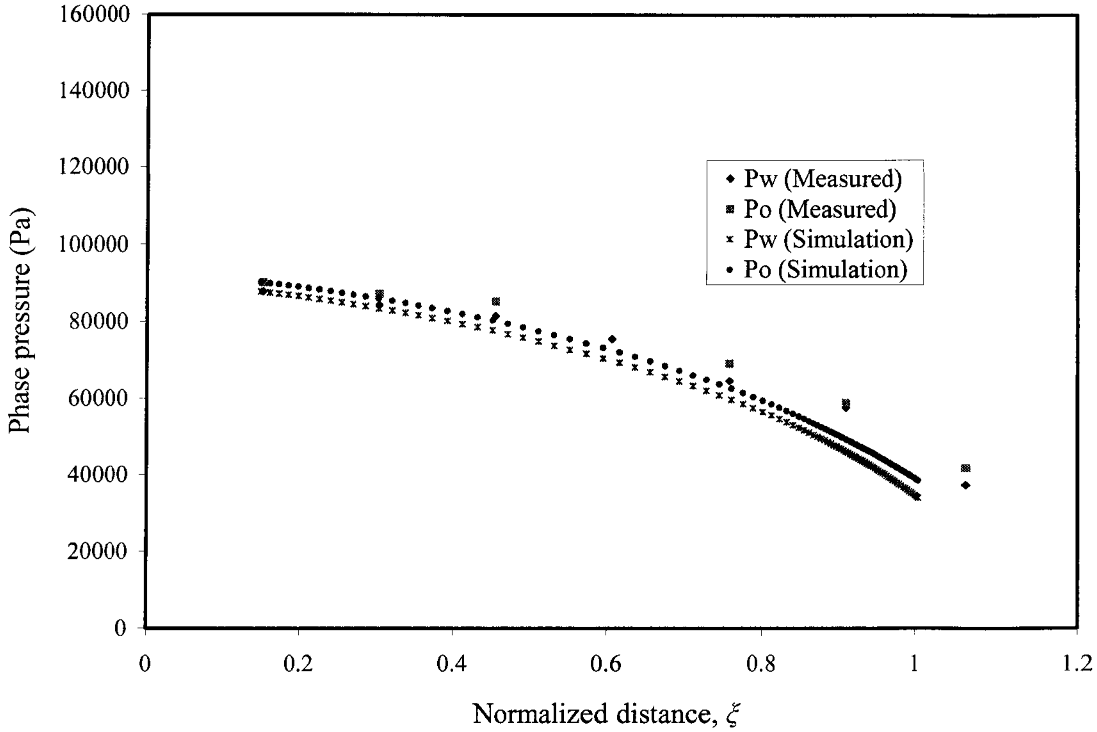


Figure 6-9: Match of pressure profiles when both viscous and capillary coupling effects are considered [Data Group B]

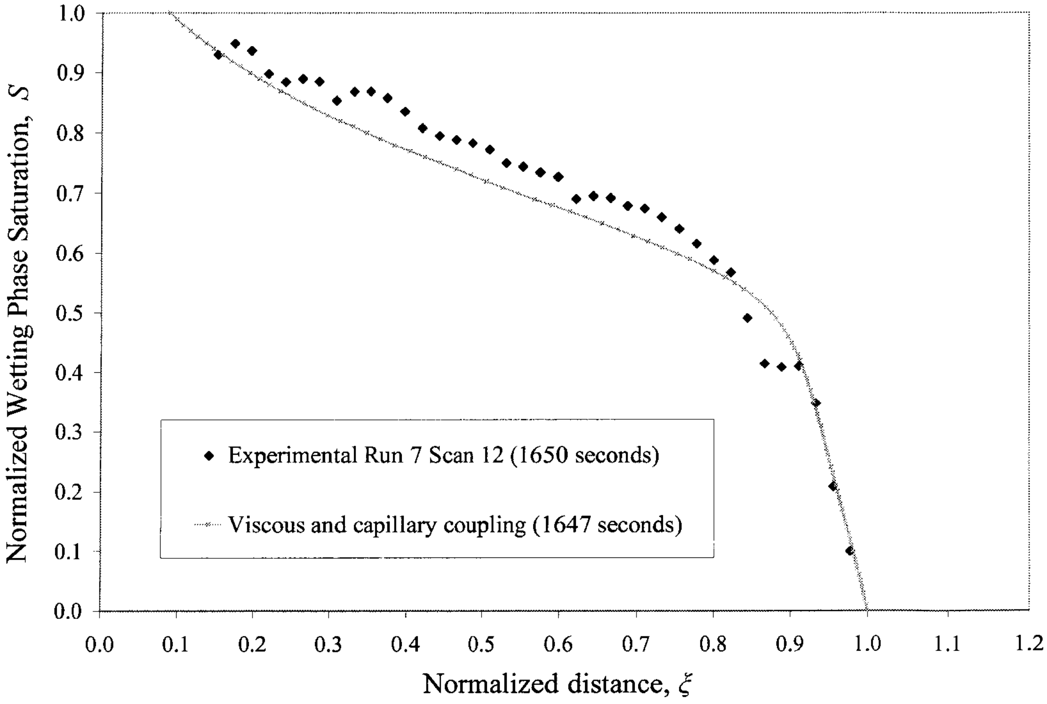


Figure 6-10: Match of saturation profiles when both viscous and capillary coupling effects are considered [Data Group C]

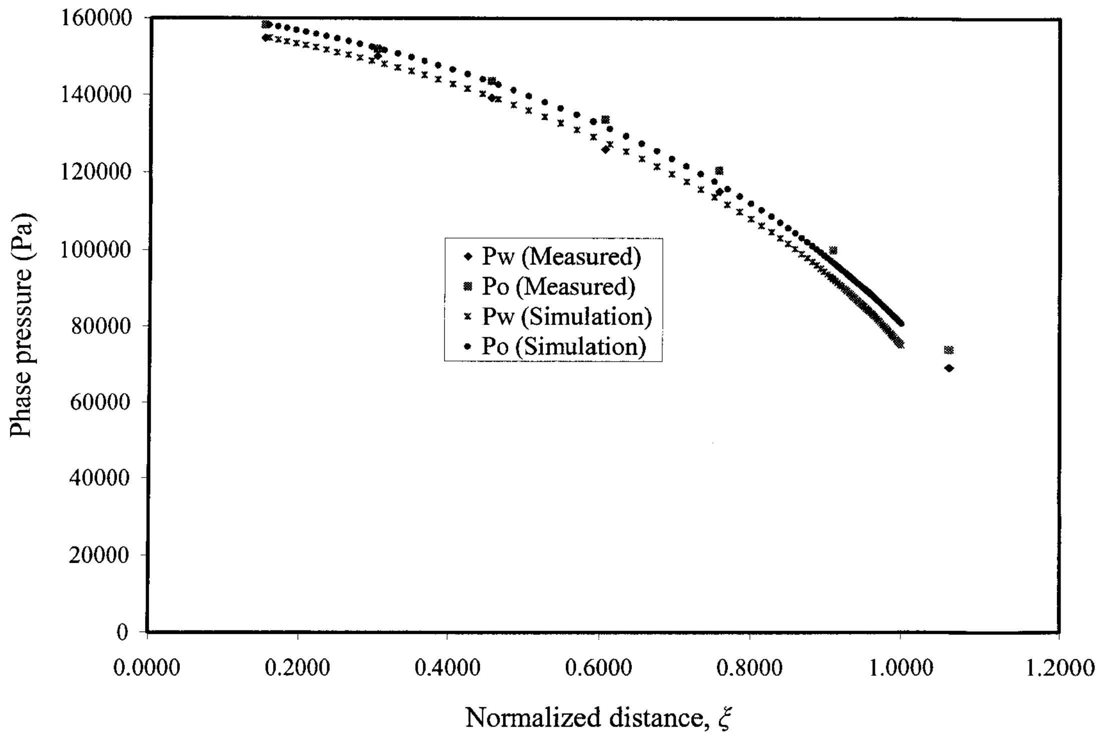


Figure 6-11: Match of pressure profiles when both viscous and capillary coupling effects are considered [Data Group C]

6.4.2 History matches based on the traditional transport equations

In order to compare the results obtained in Section 6.4.1, and to find out which model, the Modified Transport Equations model or the traditional transport equations model, gives the better history match, simulation runs based on the traditional transport equations model were carried out in two ways as well. The first way is to match the normalized injected pore volume obtained from the experiment, while the second way is to match the travel distance of the flood front. For both ways, the corresponding pressure profile matches are also given. In addition, for the sake of easy comparison, the simulation saturation profiles obtained in Section 6.4.1 are also plotted in the saturation profile matching figures presented in this section.

6.4.2.1 Matching the normalized injected pore volume

The experimental and simulation matches obtained by following the first way are shown in Figures 6-12 through 6-17.

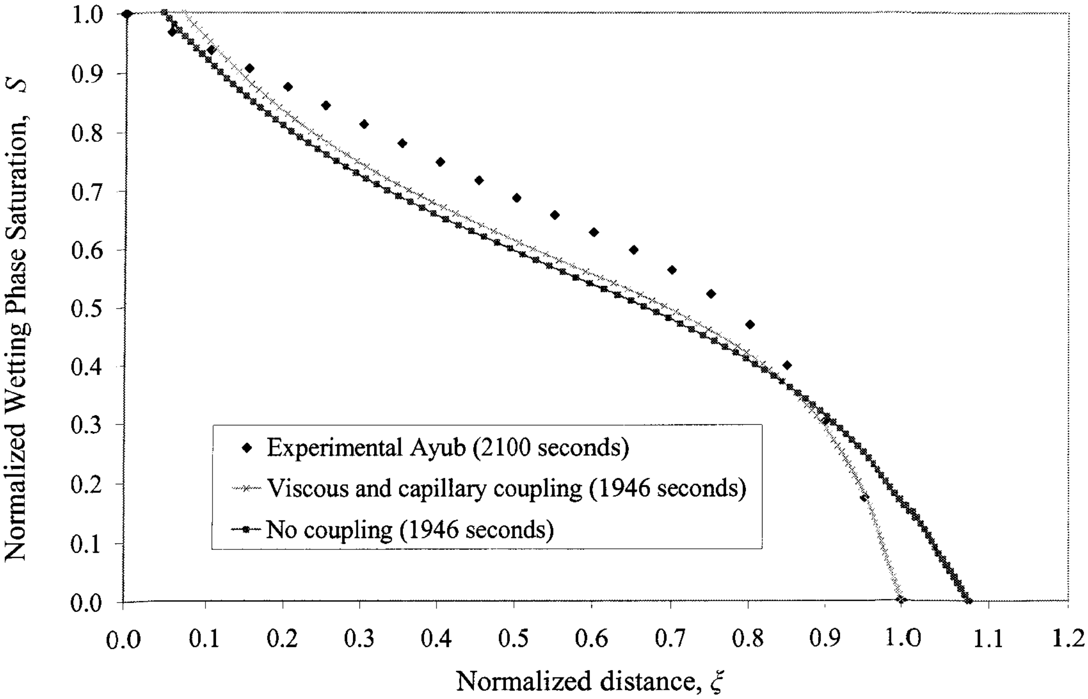


Figure 6-12: Match of saturation profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group A]

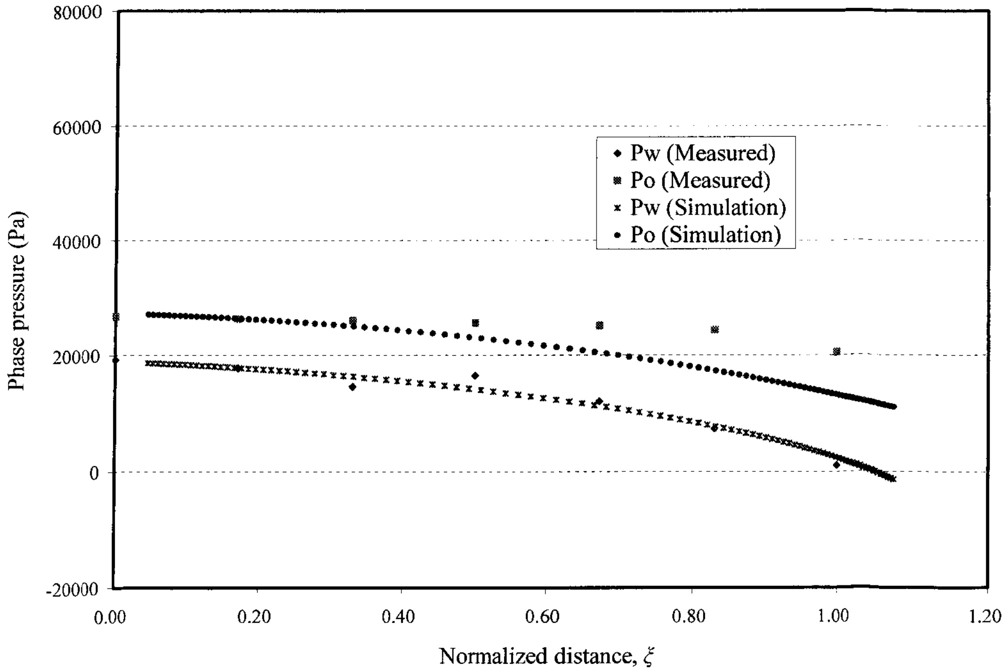


Figure 6-13: Match of pressure profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group A]

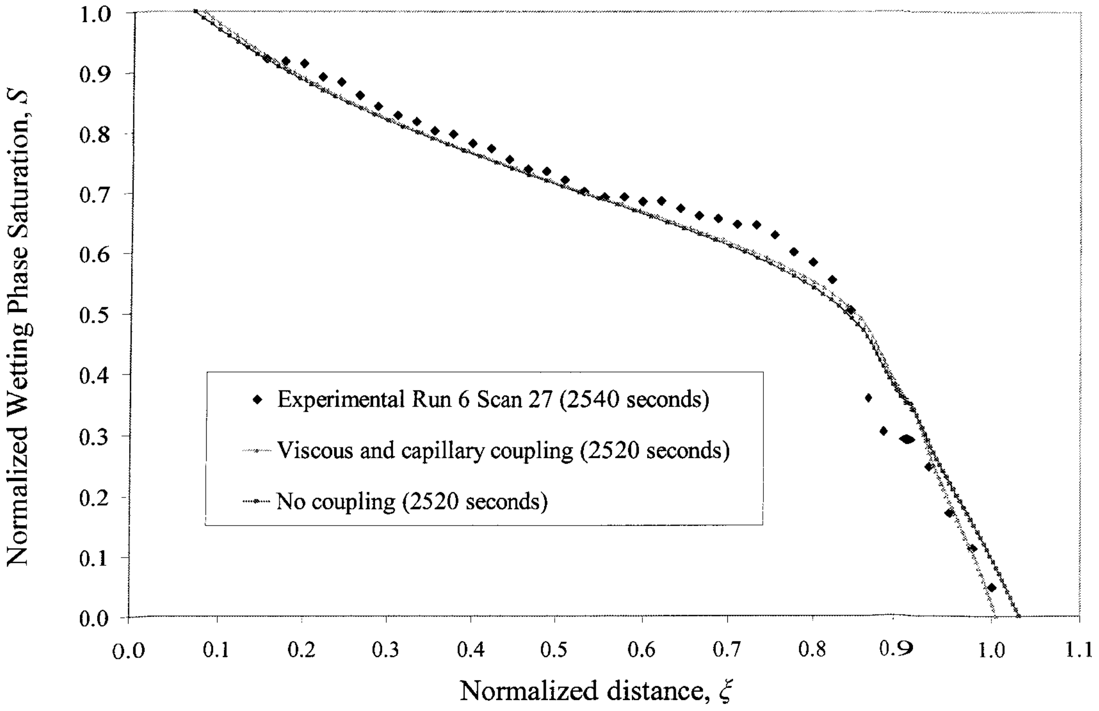


Figure 6-14: Match of saturation profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group B]

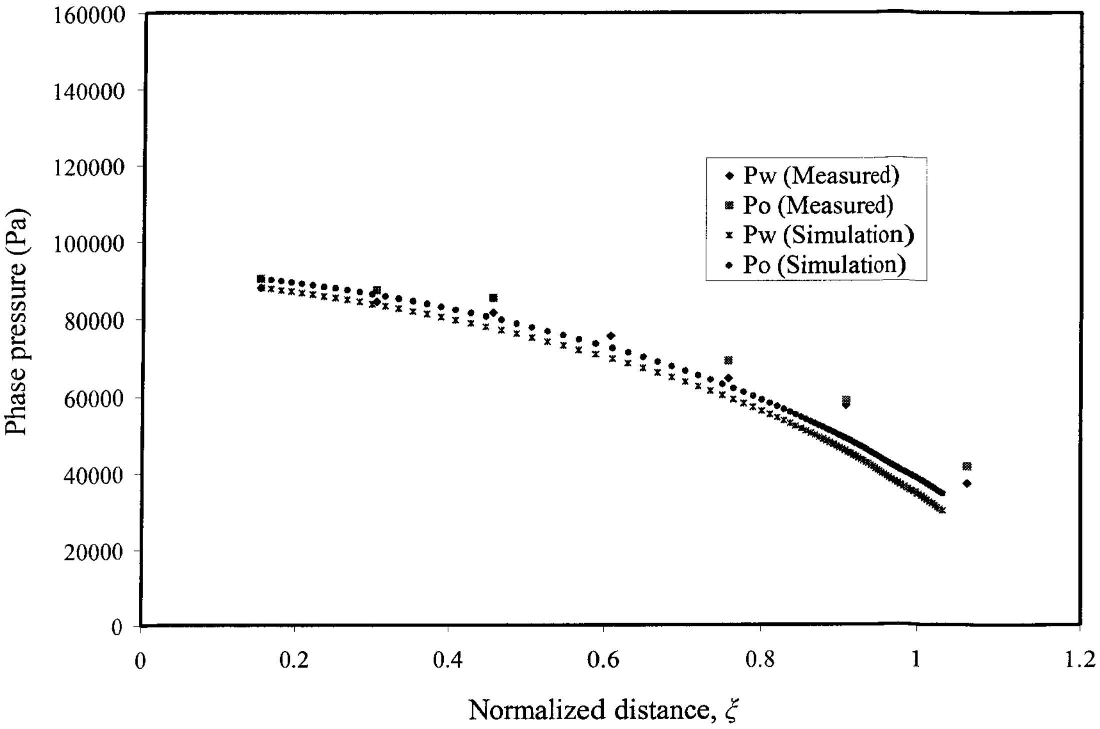


Figure 6-15: Match of pressure profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group B]

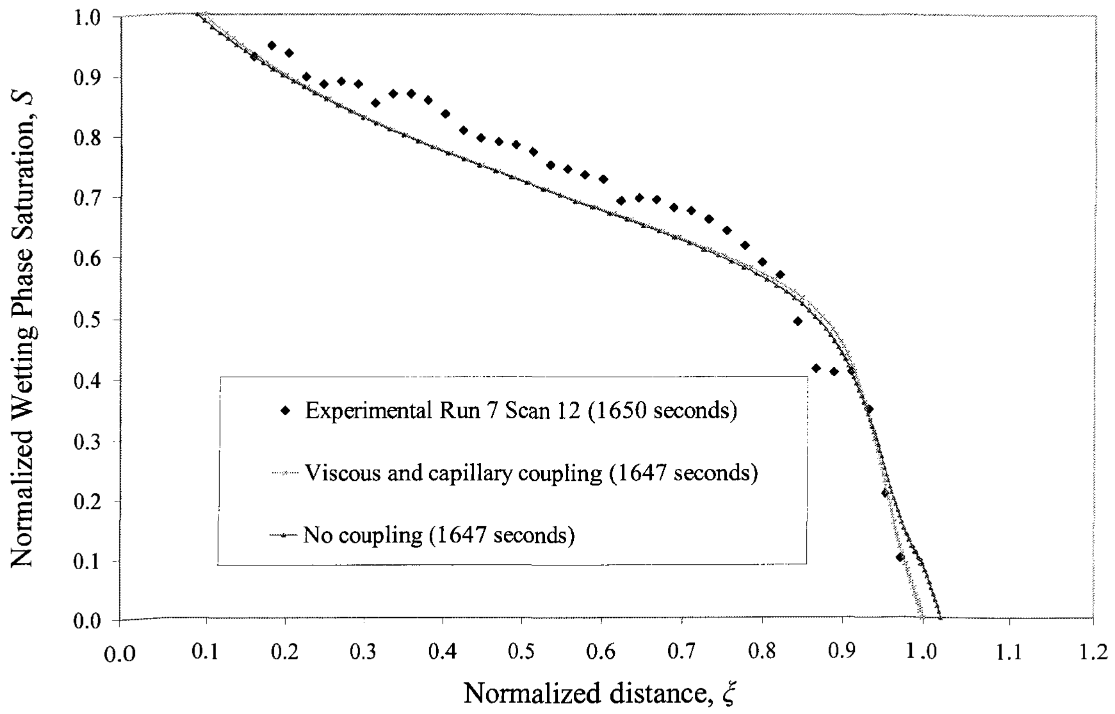


Figure 6-16: Match of saturation profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group C]

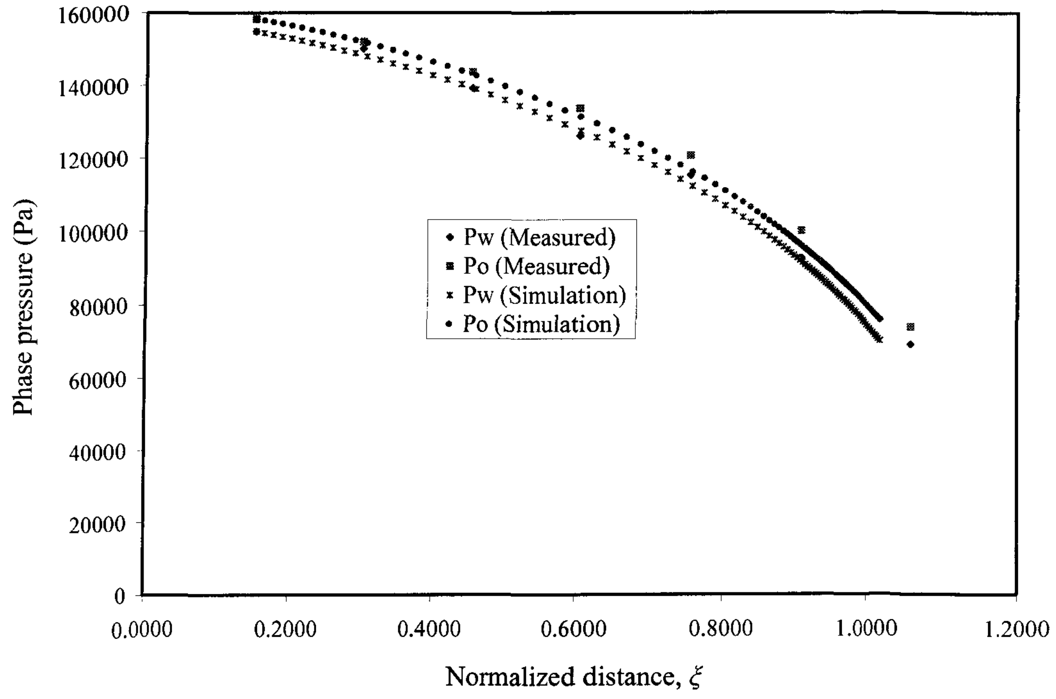


Figure 6-17: Match of pressure profiles without interfacial coupling effects considered [Matching normalized injected pore volume, Data Group C]

As can be seen in Figures 6-12, 6-14 and 6-16, the saturation profiles from simulations, with or without interfacial coupling effects considered, have the same injection time. However, their shapes are quite different. The saturation profiles from simulations without interfacial coupling effects considered have worse matches with the experimental saturation profiles than do those from simulations with interfacial coupling effects considered. These matches get worse if one increases the injection time of the saturation profiles from simulations without interfacial coupling effects considered in order to make the simulation injection time agree with that of the experiment.

By comparing Figures 6-7, 6-9 and 6-11 with Figures 6-13, 6-15 and 6-17, respectively, one can see that the shapes of the pressure profiles are essentially similar. However, it needs to be noted that, in Figure 6-13, the water phase pressures obtained from the simulations are less than zero in the vicinity of the flood front. This is physically unrealistic, and demonstrates that the use of the traditional transport equations to describe this specific experiment may be inappropriate.

6.4.2.2 Matching the travel distance of the flood front

Following the second way, history matches for the three data Groups were obtained and they are presented in Figures 6-18 through 6-23. In Figures 6-18, 6-20 and 6-22, the saturation profiles from the simulations, with or without interfacial coupling effects considered, have the same flood front travel distances. Nevertheless, the saturation profiles from simulations without interfacial coupling effects considered have smaller injection times than those from simulations with interfacial coupling effects considered. That is, the discrepancy between the injection times for experimental saturation profiles and those obtained from simulations with interfacial coupling effects considered is larger than that when no interfacial coupling effects are considered. Therefore, the matches are worse than those obtained in Section 6.4.1.

By comparing carefully the pressure profiles in Figures 6-7, 6-9 and 6-11 with those in Figures 6-19, 6-21 and 6-23, it can be seen that the matches in the latter sets of

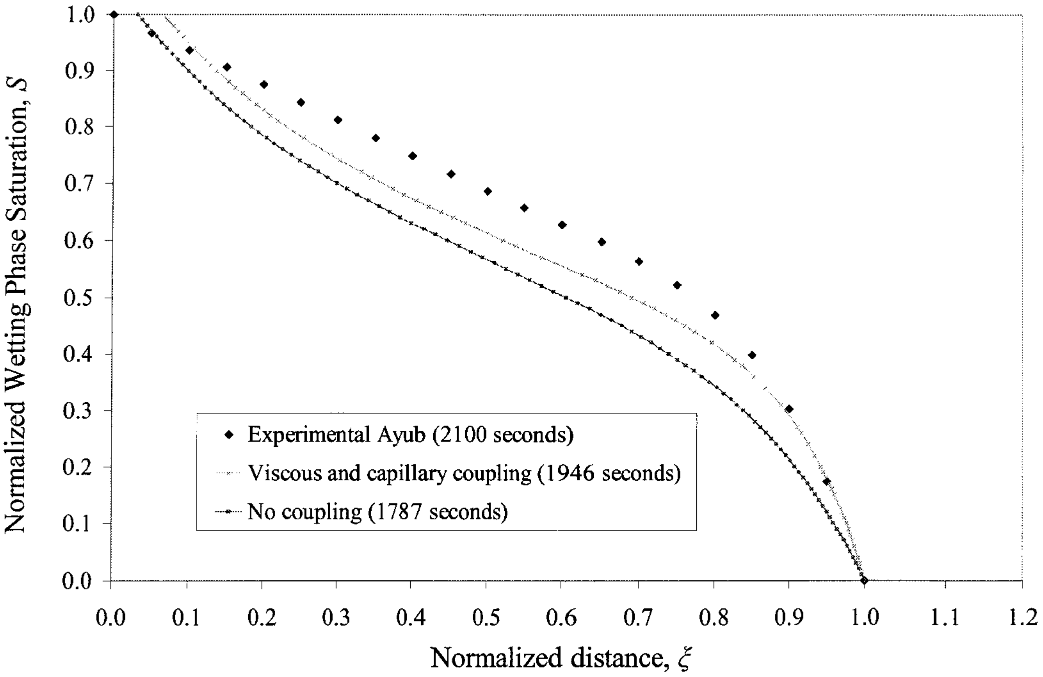


Figure 6-18: Match of saturation profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group A]

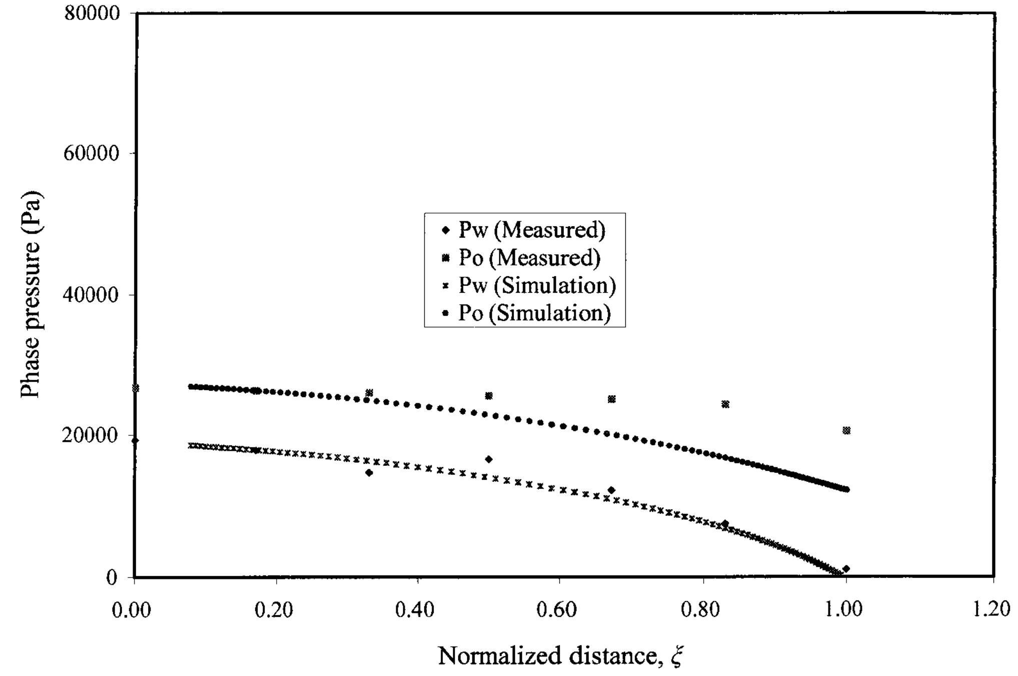


Figure 6-19: Match of pressure profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group A]

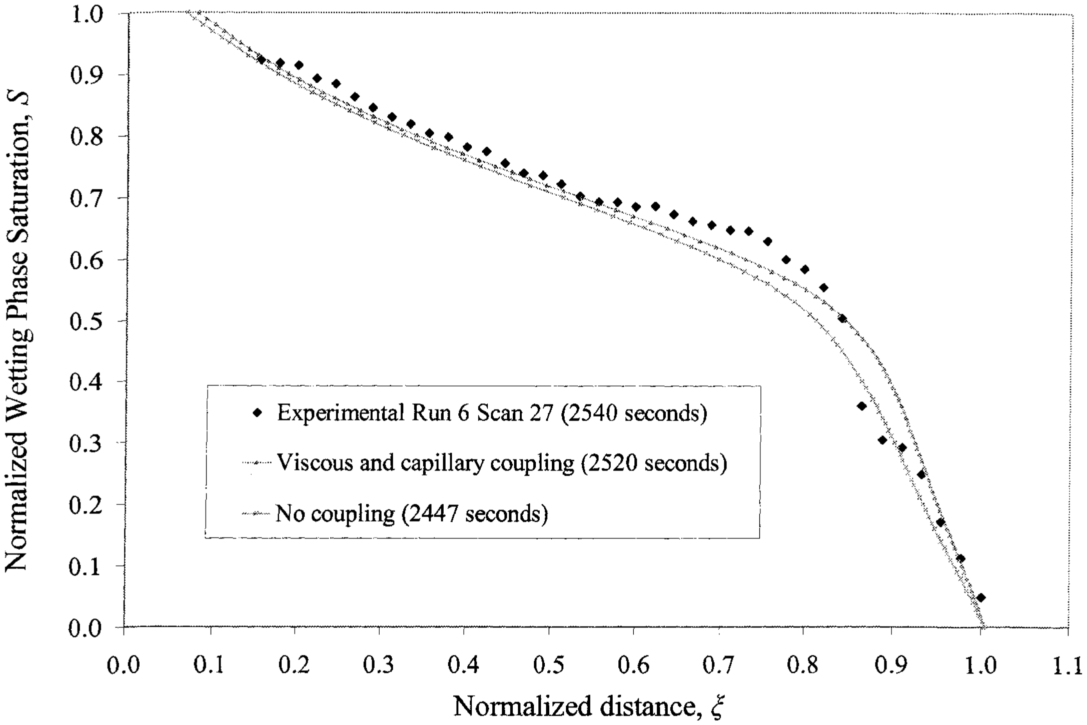


Figure 6-20: Match of saturation profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group B]

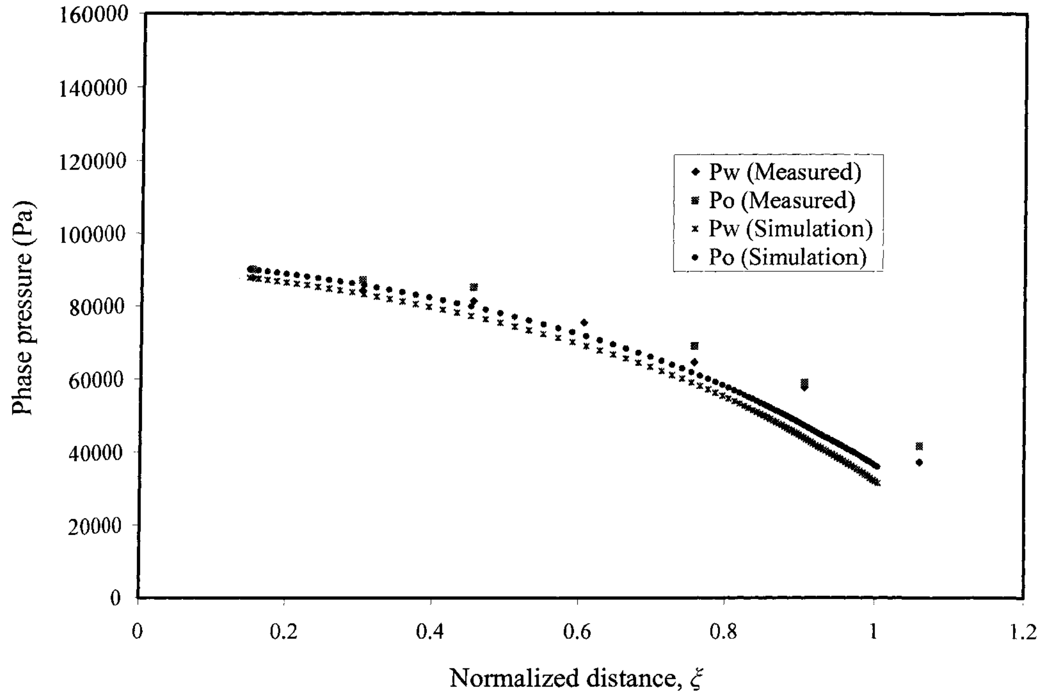


Figure 6-21: Match of pressure profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group B]

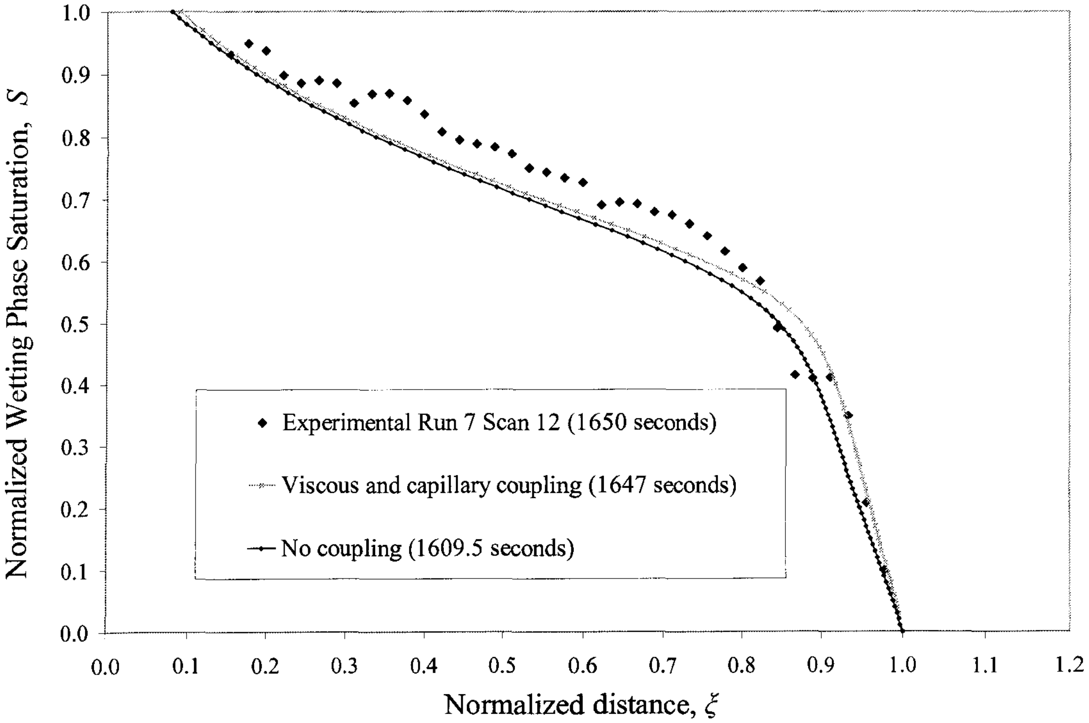


Figure 6-22: Match of saturation profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group C]

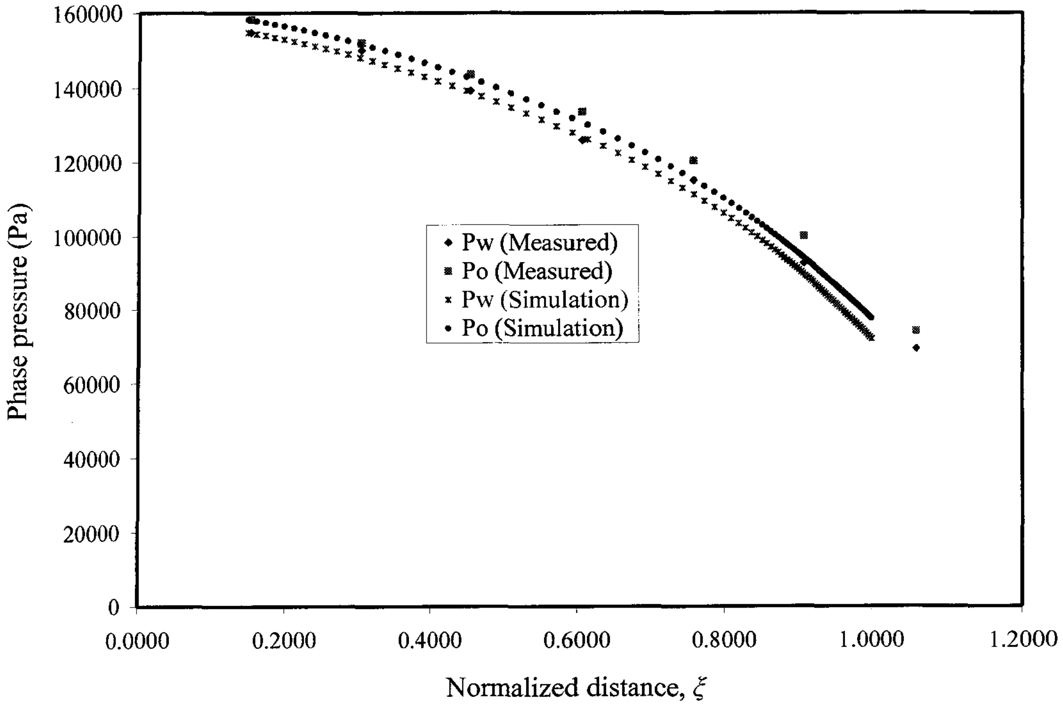


Figure 6-23: Match of pressure profiles without interfacial coupling effects considered [Matching travel distance of the flood front, Data Group C]

figures are slightly worse. However, the difference is quite small.

From what has been discussed in Section 6.4, the following conclusions can be drawn:

1. Compared to the traditional transport equations model, the Modified Transport Equations model gave better history matches for all three experimental data groups.

2. The saturation profiles were more sensitive to the influence of interfacial coupling effects than were the pressure profiles.

3. From a comparison between the saturation profiles with and without coupling effects (see Figures 6-12, 6-14, 6-16, 6-18, 6-20 and 6-22), it can be seen that that the interfacial coupling effects make the flood front steeper, resulting in a delay of water breakthrough. That is, interfacial coupling effects can increase the recovery factor at breakthrough time.

6.5 Impact of injection rate on interfacial coupling effects

As mentioned in Section 6.2, Data Groups B and C were obtained from two USCO flow experiments conducted on the same sand-fluid system with different injection rates. Therefore, these two data sets can be used to investigate the impact of injection rate on interfacial coupling effects. Based on Data Groups B and C, simulation runs, with and without interfacial coupling effects considered, were carried out, and the saturation profiles corresponding to breakthrough time are plotted in Figures 6-24 and 6-25. These two figures reveal that the higher injection rate results in a steeper flood front and a greater normalized injected pore volume at breakthrough, no matter whether the interfacial coupling effects are considered or not.

The normalized injected pore volumes achieved from both high and low injection rates with and without interfacial coupling effects considered, were tabulated and are compared in Table 6-3. As can be seen from this table, the interfacial coupling effects have less impact on the higher injection rate case. Therefore, it can be concluded that

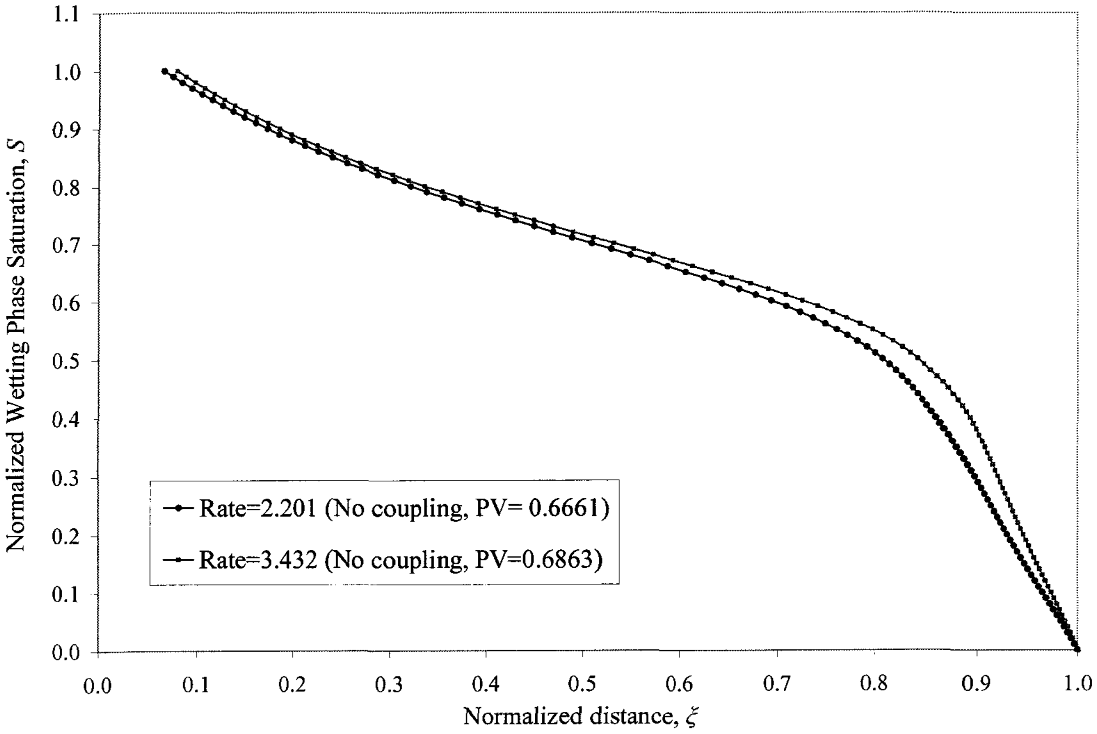


Figure 6-24: Saturation profiles at breakthrough time for different injection rates [Without interfacial coupling effects considered; Data Groups B and C]

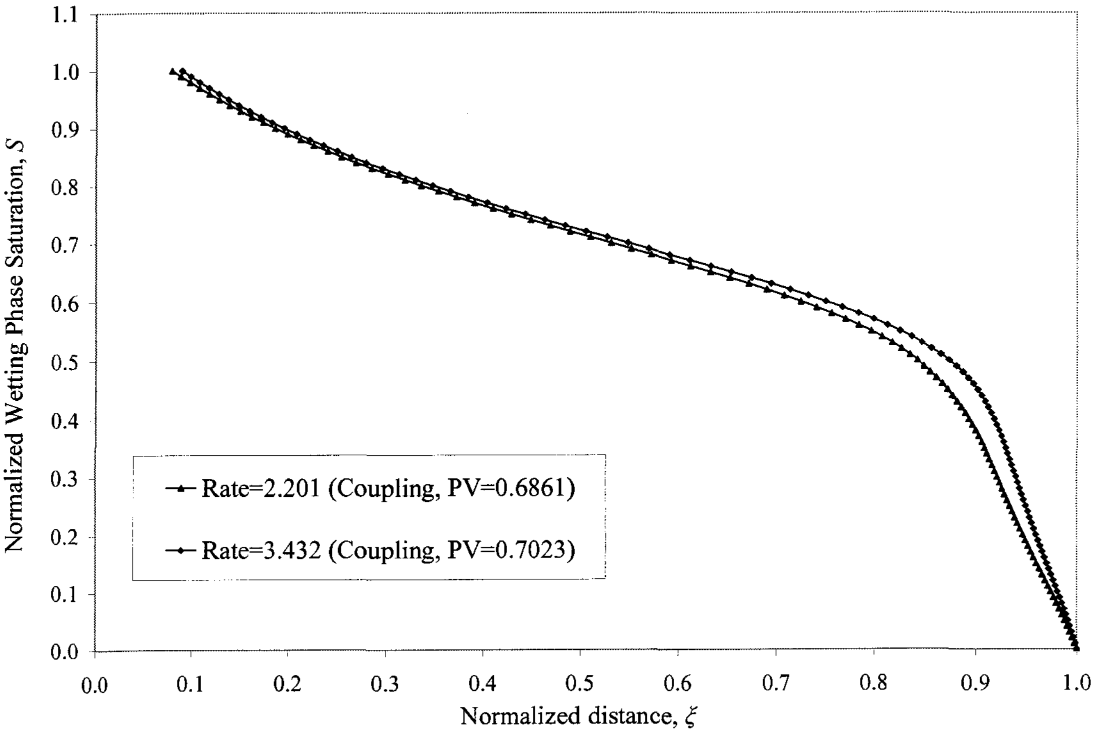


Figure 6-25: Saturation profiles at breakthrough time for different injection rates [With interfacial coupling effects considered; Data Groups B and C]

Injection rate	Injected PV at breakthrough		Difference	
(E-08 m^3/s)	both coupling	no coupling	(PV*-PV)	(PV*-PV)/PV
2.201	0.6861	0.6661	0.020	3.0%
3.432	0.7023	0.6863	0.016	2.3%

PV*=Injected PV with some effect
PV=Injected PV without any effect

Table 6-3: Impact of injection rate on interfacial coupling effects

the higher the injection rate, the steeper the flood front and the less influence the interfacial coupling effects have.

6.6 Effect of reservoir angle of dip on interfacial coupling effects

In order to observe clearly the effect of the reservoir angle of dip on the interfacial coupling effects, the density difference between the two fluids has to be large. In this regard, Data Group A was modified so that the oil density was reduced to 500 from 830 Kg/m^3 and the water density was increased to 1000 from 990 Kg/m^3. On the basis of the modified Data Group A, simulation runs, with and without interfacial coupling, were performed for various reservoir angles of dip. The saturation profiles corresponding to breakthrough time are presented in Figures 6-26 and 6-27, and the breakthrough time for various cases are listed and analyzed in Table 6-4.

As can be seen from Figures 6-26 and 6-27, no matter whether the interfacial coupling effects are considered or not, as the reservoir angle of dip increases, the saturation profile gets steeper, and the breakthrough time becomes larger. However, as shown in Table 6-4, the influence of the interfacial coupling effects decreases as the reservoir angle of dip increases. In summary, for a certain sand-fluid system, the higher the reservoir angle of dip, the larger the breakthrough time and the less the influence of the interfacial coupling effects.

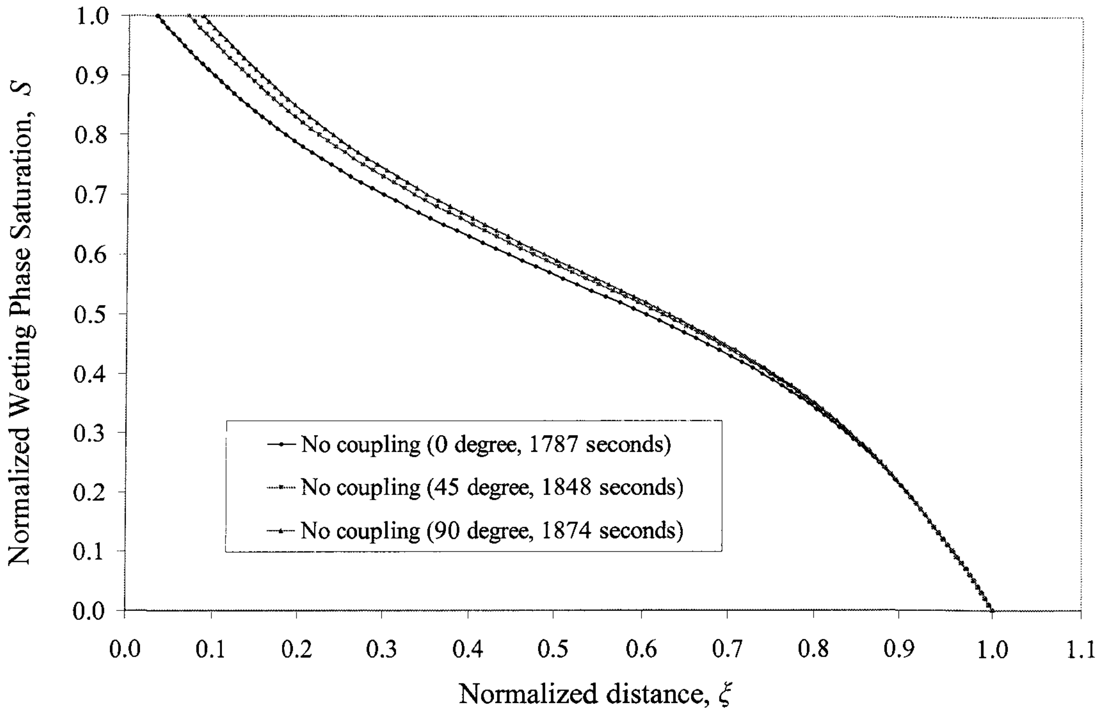


Figure 6-26: Saturation profiles at breakthrough time for different reservoir dipping angles [Without interfacial coupling effects considered; Modified Data Group A]

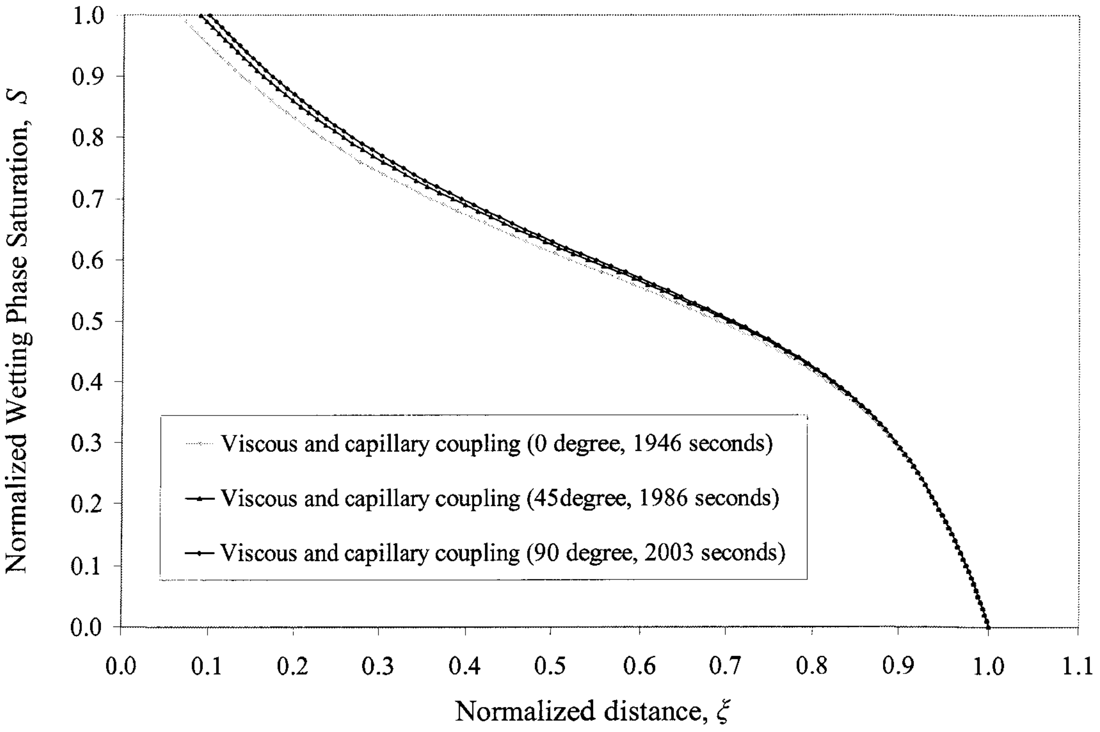


Figure 6-27: Saturation profiles at breakthrough time for different reservoir dipping angles [With interfacial coupling effects considered; Modified Data Group A]

Reservoir angle of dip degrees	breakthrough time (seconds)		difference	
	Both coupling	No coupling	seconds	(both-no)/no
0	1946	1787	159	8.9%
45	1986	1848	138	7.5%
90	2003	1874	129	6.9%

Table 6-4: Impact of reservoir dipping angle on interfacial coupling effects

6.7 Influence of hydrodynamic, viscous and capillary coupling factors

On the basis of Data Group A, the influence of hydrodynamic, viscous and capillary coupling factors were analyzed individually. The analyses were achieved by carrying out simulation runs with the corresponding control option in the GUI (see Figure 5-2 for details) selected. As proposed by Bentsen and Manai (1991, 1993), the hydrodynamic factor R_{12} is defined by the following equation:

$$R_{12} = 1 - a*(1-S), \qquad (6.1)$$

where, the coefficient, $a*$, must be determined experimentally. Because no information concerning the parameter $a*$ was available, its value was assumed, for sensitivity analysis purposes, to be 0.05, a value determined by Bentsen and Manai (1991, 1993) in a similar sand-fluid system. As was the case in Section 6.4, the viscous coupling value was set as 2 during the simulation runs.

Comparisons of the saturation profiles with and without coupling or hydrodynamic effects are given in Figures 6-28 through 6-30. The normalized injected pore volumes corresponding to each saturation profile were analyzed and are given in Table 6-5.

As shown in Figure 6-28, the saturation profiles with and without hydrodynamic effects are close to one another. The corresponding injected normalized pore volumes were 0.5647 and 0.5581, respectively. The injected normalized pore volume at breakthrough increases by 0.0066, about 1.18%, when hydrodynamic effects ($a*$=0.05) were considered.

As can be seen in Figure 6-29, the saturation profiles without any coupling effect, and

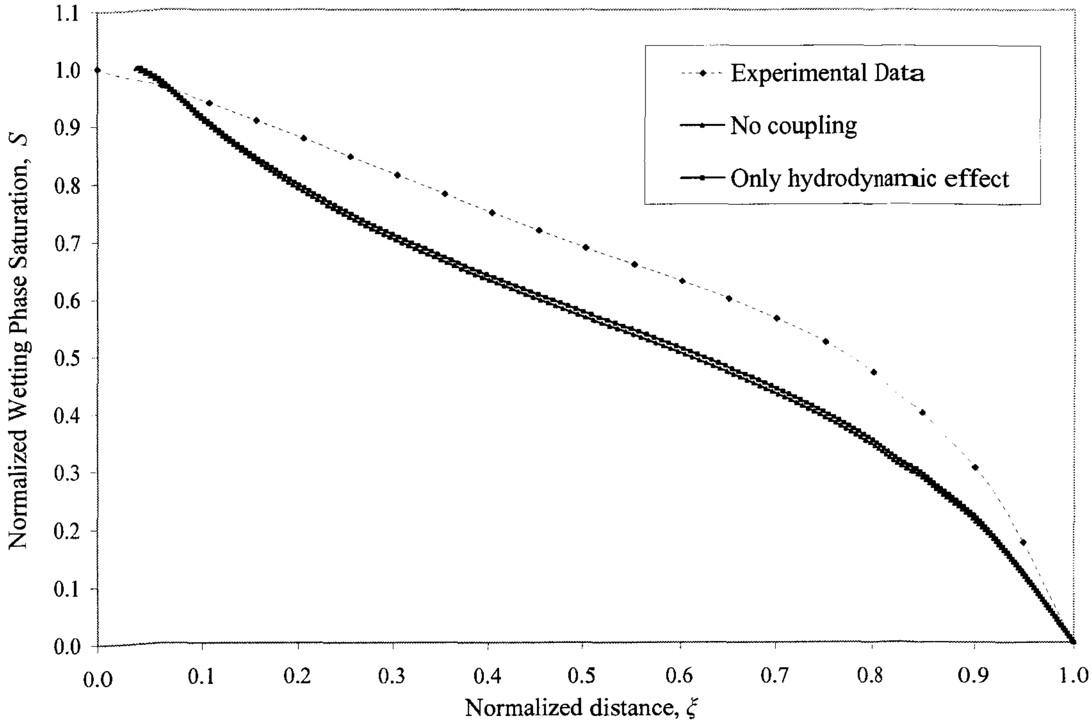


Figure 6-28: Comparison of saturation profiles with or without hydrodynamic effects

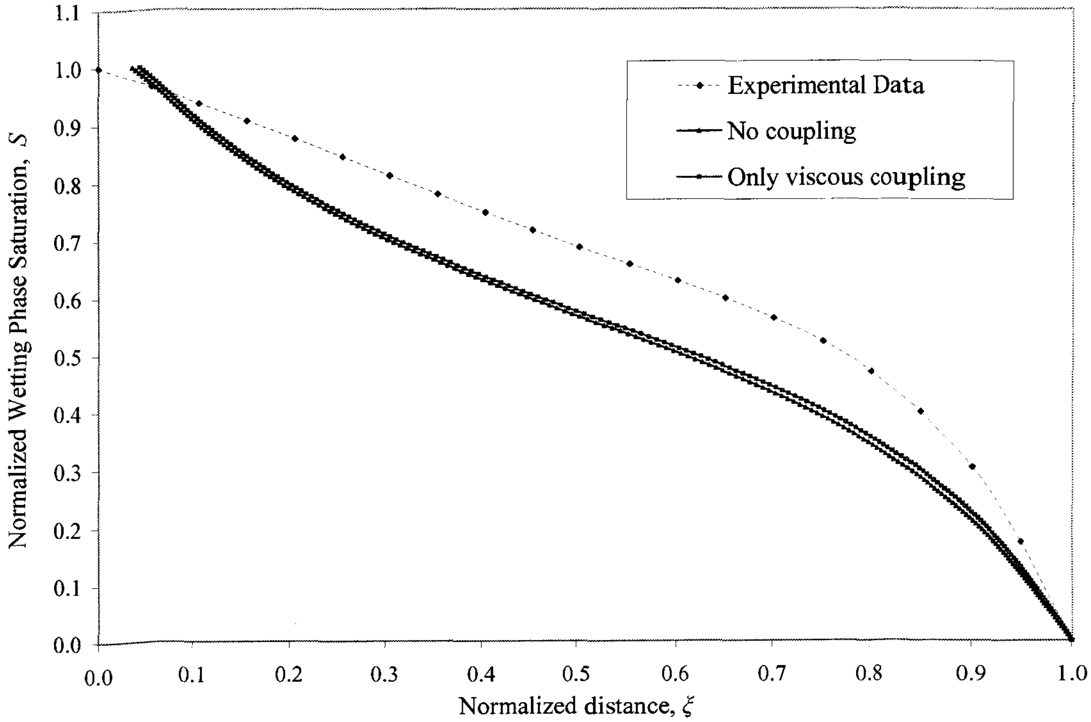


Figure 6-29: Comparison of saturation profiles with or without viscous coupling

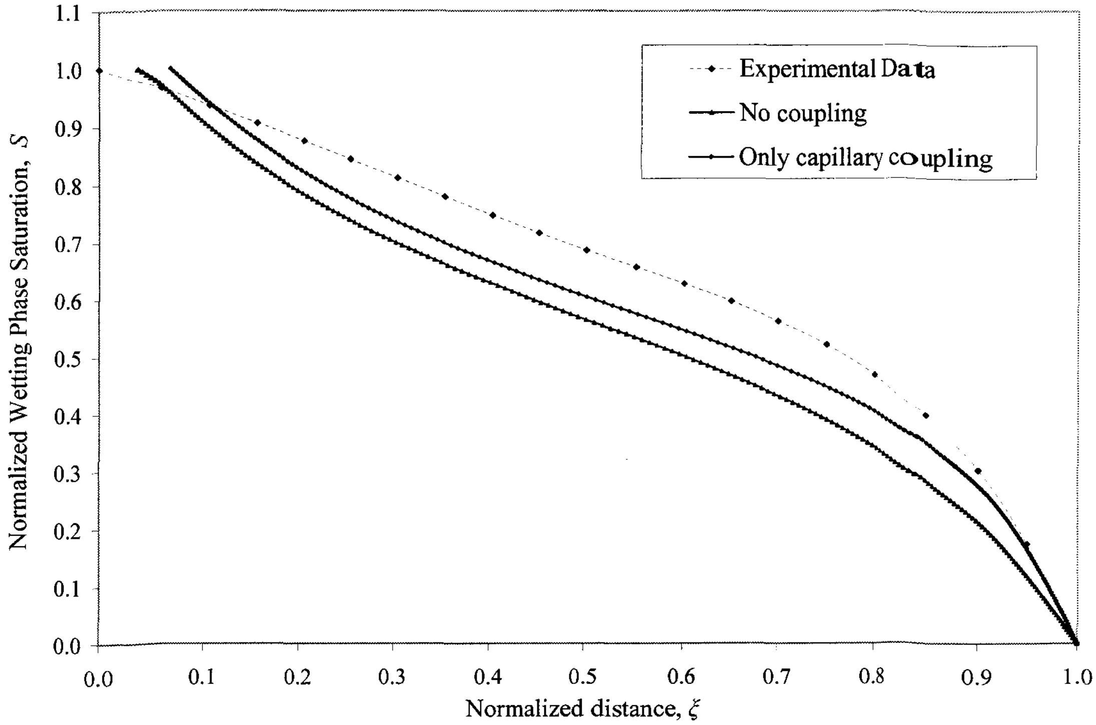


Figure 6-30: Comparison of saturation profiles with or without capillary coupling

Item	Injected PV at breakthrough		(PV*-PV)/PV	
	$M_r = 5.31$	$M_r = 3.18$	$M_r = 5.31$	$M_r = 3.18$
No effects	0.5581	0.6478		
Hydrodynamic effect	0.5647	0.6531	1.18%	0.82%
Viscous coupling effect	0.5669	0.6602	1.58%	1.91%
Capillary coupling effect	0.6007	0.7022	7.63%	8.40%

PV*=Injected PV with some effect
PV=Injected PV without any effect

Table 6-5: Injected pore volume at breakthrough under different conditions

with maximum viscous coupling effect, have essentially similar shapes. Consideration of the viscous coupling effect resulted in an increase of 0.0088 (1.58%) in the normalized pore volume at breakthrough, as compared to when this effect was neglected.

Unlike viscous coupling, capillary coupling has a large effect on the saturation profile, as can be seen in Figure 6-30. The area under the saturation profile with capillary coupling is 0.0426 or 7.63% greater than that without any effect.

6.8 The role of mobility ratio on coupling and hydrodynamic effects

To see the effect of mobility ratio on coupling and hydrodynamic effects, the non-wetting phase viscosity of Data Group A was reduced to 0.009 Pa.s from 0.015 Pa.s, while the other data were kept unchanged, which results in a decrease in the mobility ratio (M_r) from 5.31 to 3.18. Using the modified Data Group A, the simulation was run in the same manner as described in Section 6.7. The simulation results were analyzed, and the results of the analysis are listed in Table 6-5.

As shown in these results, a more favorable mobility ratio leads to an increase in the normalized injected pore volume at breakthrough, no matter which coupling or hydrodynamic effects were considered. Moreover, the viscous and capillary coupling effects both have more impact in the more favorable mobility ratio case than in the less favorable mobility ratio case. On the contrary, the hydrodynamic effect has less impact in the more favorable mobility ratio case.

6.9 Interrelated parametric equations

For a given sand-fluid system, it is thought that the capillary pressure and the relative permeability curves should be related. However, the way in which these curves are related is not known. In this regard, it has been found that if the relative permeability and the capillary pressure are not measured accurately, or correctly, problems can arise in the simulation analysis. At first glance, the flood front of the saturation profile, presented in Figure 6-10, which was obtained by simulation, appears to be quite smooth. However, as can be seen in Figure 6-31, there is a small change in curvature, around the flood front, of the curve $\partial\xi/\partial S$ versus S. This is an indication of the inaccuracy of the experimental data. However, it is hard to tell which parameter causes the problem, because $\partial\xi/\partial S$ is determined by so many parameters, as can be seen in Equation (6.2):

$$\frac{\partial\xi}{\partial S}=-\frac{N_c C(S)}{f(S,\tau)-G(S)}\ . \qquad (6.2)$$

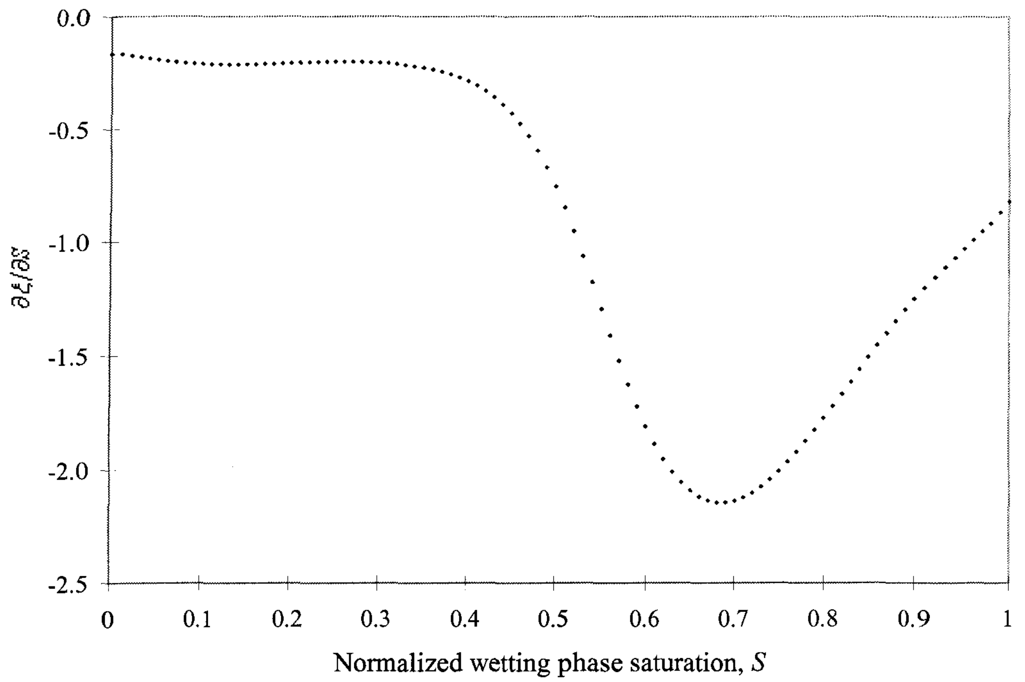


Figure 6-31: The first derivative of normalized distance with respect to saturation

During trial and error analysis, it was found that the curvature of $\partial\xi/\partial S$ versus S around the flood front was very sensitive to the value of dk_{r1}/dS near S=0. With a decrease in the value of dk_{r1}/dS near S=0, the curvature of $\partial\xi/\partial S$ versus S around the flood front becomes smaller until the reversal in curvature disappears. In contrast, an increase in the value of dk_{r1}/dS near S=0 can cause a foot on the flood front. For example, if the fitting coefficients listed in Table 6-6 are used to fit the relative permeability data for Data Groups B and C, one obtains the fitted curves shown in Figure 6-32. By comparing Figures 6-32 and 6-1, it can be seen that the curve shapes

Wetting phase		Non-wetting phase	
a_1	0.2820	a_2	0.9765
b_1	-0.3892	b_2	0.0110
c_1	0.5072	c_2	0.0125
d_1	0.6	d_2	0
$a_1+b_1+c_1+d_1$	1.0000	$a_2+b_2+c_2+d_2$	1.0000
Sum(Error^2)	5.59E-05	Sum(Error^2)	1.37E-07

Table 6-6: Fitted coefficients for k_{r1} and k_{r2}

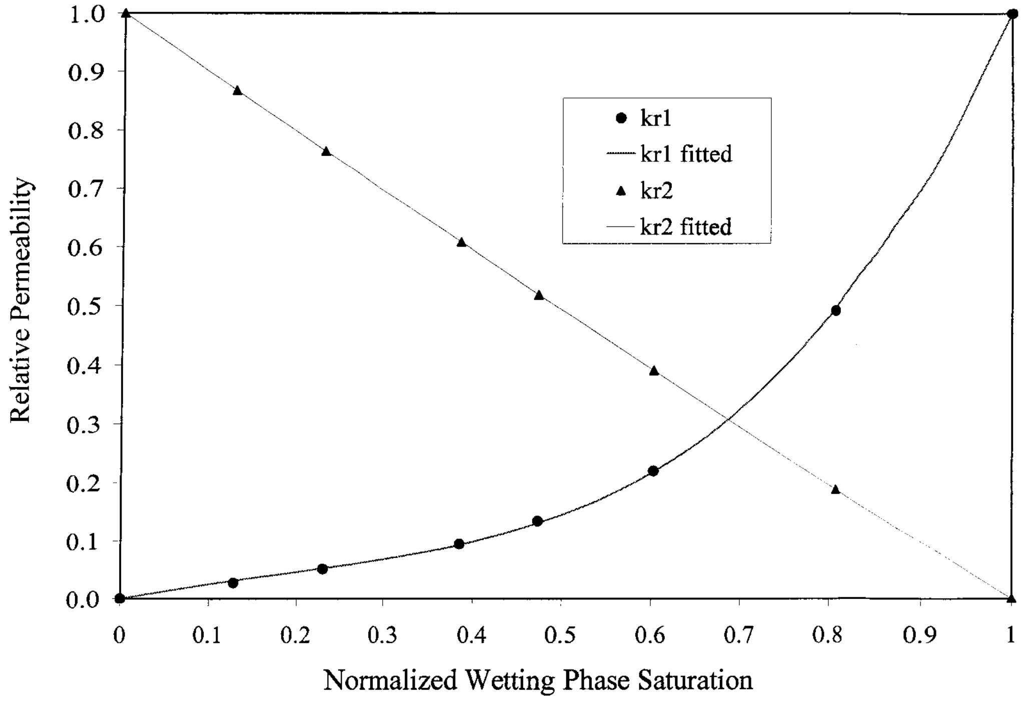


Figure 6-32: Fitted relative permeability curves

in Figure 6-32 are quite similar to those in Figure 6-1. However, the simulation results are quite different as can be seen by comparing Figures 6-10 and 6-33. Note

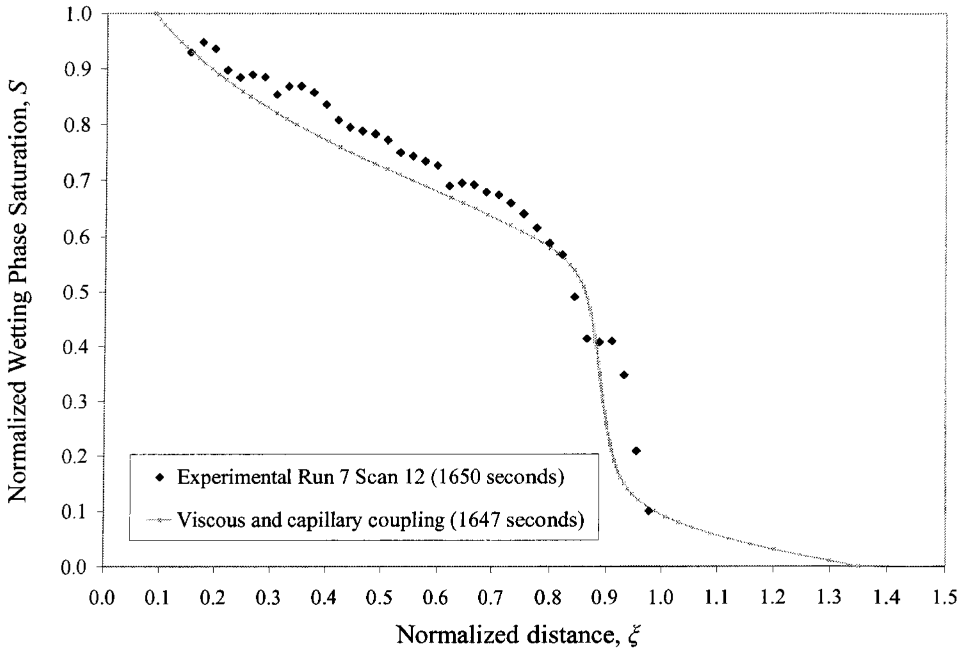


Figure 6-33: Match of saturation profiles when both viscous and capillary coupling effects are considered [Data Group C]

particularly that there is a foot on the flood front in Figure 6-33, which does not appear in Figure 6-10.

Again, because $\partial\xi/\partial S$ is determined using Equation (6.2), it is believed that the change in curvature on the curve $\partial\xi/\partial S$ versus *S*, around the flood front, can be made to disappear by modifying the parameters (k_{r1}, k_{r2}, P_c, an so on) on the RHS of Equation (6.2), rather than by modifying only dk_{r1}/dS. However, how much each parameter should be modified to get the right solution is unknown, because the relationship between the parameters is unknown. Moreover, even if an adjustment is made only to dk_{r1}/dS, it is still hard to determine which value of dk_{r1}/dS gives rise to the correct simulation result. Therefore, it is extremely important to have high quality data. In addition, care must be taken when selecting the fitting equations for the relative permeability and capillary pressure curves.

6.10 k_{r1} @ S=0 and the behavior of the flood front

As discussed in Section 6.9, the curvature of $\partial\xi/\partial S$ versus *S* around the flood front was very sensitive to the value of dk_{r1}/dS near *S*=0. This phenomenon can be explained by the following mathematical analysis.

From Equation (5.27), it can be seen that the propagation velocity of the saturation profile depends on the value of $\partial f/\partial S$. Near *S*=0, $\partial f/\partial S$ is determined by Equation (6.3) [the derivation of this equation has been carried out by Dr. Bentsen and can be found in Appendix E].

$$\lim_{S\to 0}\frac{\partial f}{\partial S}=M_r\frac{dk_{r1}}{dS}+N_c\frac{d\pi_c}{dS}\frac{dk_{r1}}{dS}\frac{\partial S}{\partial \xi}. \tag{6.3}$$

Collecting items on the RHS of Equation (6.3), one obtains:

$$\lim_{S\to 0}\frac{\partial f}{\partial S}=\left(M_r+N_c\frac{d\pi_c}{dS}\frac{\partial S}{\partial \xi}\right)\frac{dk_{r1}}{dS}. \tag{6.4}$$

As can seen from Equation (6.4), the extent of the effect of dk_{r1}/dS near S=0 on the flood front propagation velocity, and hence the curvature of $\partial\xi/\partial S$ versus S around the flood front, depends on the magnitude of the items within the bracket on the RHS of this equation. That is, if the magnitude of these items is large, small errors in dk_{r1}/dS will make a big difference in the flood front propagation velocity, and vice versa.

Moreover, it needs to be noted that the proper choice of the parametric fitting equations for k_{r1} and P_c is extremely important. The selected parametric fitting equations have to satisfy certain conditions, which are (a) $C(S)$ [see Equation (5.30)] must equal zero at S=0, and (b) $\frac{d\pi_c}{dS}\frac{dk_{r1}}{dS}$ must have a reasonable finite value [Shen and Ruth (1994) noted this requirement, and they specified this requirement in a different way]. Otherwise, either the basic concept will be violated or the flood front near S=0 will travel at a physically unrealistic high velocity [see Equation (6.3)].

CHAPTER 7

Conclusions and recommendations

In this study, interfacial coupling phenomena in two-phase flow through porous media were tested by using experimental and numerical simulation methods. On the basis of the work that has been done, several conclusions have been drawn, and some recommendations have been given as well.

7.1 Conclusions for interfacial coupling effects

1. Compared to the traditional transport equations model, the Modified Transport Equations model gave better history matches to the experimental data groups used in this study.
2. Interfacial coupling does play a role in two-phase flow through porous media, even in the cocurrent flow case. Interfacial coupling effects make the flood front steeper, and hence delay water breakthrough. In other words, interfacial coupling effects can increase the recovery factor at water breakthrough. Without considering this effect, that is, if the traditional transport equations model is used to simulate the petroleum reservoir recovery process, the recovery factor at breakthrough will be underestimated.
3. For a given water flooding reservoir, interfacial coupling effects decrease as the injection rate increases.
4. For a given sand-fluid system, the higher the reservoir angle of dip, the larger the breakthrough time, and the smaller the influence of interfacial coupling effects.
5. The capillary coupling effect is more significant than the hydrodynamic and viscous coupling effects.

6. In more favorable mobility ratio cases, the viscous and capillary coupling effects have more impact on recovery at breakthrough, while the hydrodynamic effect has less impact.

7. Saturation profiles are more sensitive than pressure profiles to the influence of interfacial coupling effects.

7.2 Conclusions for interfacial coupling simulator

1. Based on the Modified Transport Equations and on the Interfacial Coupling Simulator developed by Ayodele (2004), Interfacial Coupling Simulator (2.0) was developed and validated theoretically and experimentally in this study.

2. Compared to the Interfacial Coupling Simulator developed by Ayodele (2004), in Interfacial Coupling Simulator (2.0), (a) the variable inlet saturation conditions were employed by using the material balance check technique, (b) both the Newton-Raphson solver and the Newton-Jacobi solver were recoded and can work properly.

3. Interfacial Coupling Simulator (2.0) can be used to analyze core flooding experiments with or without considering hydrodynamic and interfacial coupling effects.

4. The Java$^{\text{TM}}$ programming language was used when developing Interfacial Coupling Simulator (2.0). Therefore, this simulator, without any modification, can run on any platform in which a Java$^{\text{TM}}$ virtual machine (JVM) is available.

7.3 Recommendations

1. Interfacial Coupling Simulator (2.0) was built on the basis of Bentsen's equation, a Lagrangian form equation which uses the fractional flow concept. Therefore, this simulator can not be used to analyze the process of countercurrent flow through porous media for the following two reasons: (a) a Lagrangian form equation can not take care of the outlet end boundary conditions that are required

to simulate countercurrent flow; (b) the fractional flow concept is not valid for the countercurrent flow process. In order to see if the second term in the big bracket on the right hand side (RHS) of Equations (3.35) and (3.36) is organized properly or not, that is, in order to be able to test the Modified Transport Equations under countercurrent flow situations, it is recommended to develop, on the basis of the Modified Transport Equations, a numerical simulator which can handle countercurrent flow.

2. During experiments, it was found that the looseness of the driving chain of the saturation measurement system can cause vibration of the core, and hence influence the accuracy of the measured frequency and saturation. Therefore, it is recommended to take the frequency measurements when the saturation measurement sensor is moving from the inlet end towards the outlet end of the core, because, in this direction, the driving chain is relatively tight. This approach became possible after the data acquisition system was upgraded. Moreover, it is better to calibrate the saturation measurement sensor at each frequency sampling location along the core.

3. In order to achieve more accurate water phase pressure measurements, three suggestions are provided. (a) Search for a filter material that has a higher permeability to water and a higher threshold pressure to oil than does the material that was used in this study; (b) use a high porosity water wet disc as the support base for the hydrophilic membrane; (c) ensure that there is no air trapped in the filter material and in the cavity of the pressure transducers by drawing a vacuum on them and then filling them with water.

4. Regarding the data acquisition system, it is suggested that the UPC-L interface card with an ISA bus be replaced by a UPC-L interface card with a PCI bus, once it is available, so that the old computer can be removed from this data acquisition system, and hence the operation of this system can be made much easier.

References

- Adisoemarta, P.S., (2000). Detecting changes in shale water content through changes in I dissipation factor. Petroleum Society paper number 2000-32 presented at the petroleum Society's international petroleum conference, Calgary, Alberta, Canada, 4-8 June, 2000.

- Ayodele, O.R., (2004). Mathematical modelling, experimental testing and numerical simulation of interfacial coupling phenomena of two-phase flow in porous media. PhD Thesis, University of Alberta, Edmonton, Canada, 2004.

- Ayub, M., (2000). Experimental testing of interfacial coupling phenomena in two-phase flow. PhD Thesis, University of Alberta, Edmonton, Canada, 2000.

- Ayub, M. and Bentsen, R.G., (2000). Measurement of dynamic saturation profiles. JCPT. September 2000, 39(9), 54-61.

- Ayub, M. and Bentsen, R.G., (1999). Interfacial viscous coupling: a myth or reality? J. Petrol. Sci. Eng. 23,13-26.

- Babchin, A. and Yuan, J., (1997). On the capillary coupling between two phases in a droplet train model. Transport in Porous Media. 26, 225-228.

- Bear, J., (1972). Dynamics of fluids in porous media. American Elsevier Publishing Company, Inc., New York.

- Bear, J. and Bachmat, Y., (1991). Introduction to Modeling Phenomena of Transport in Porous Media. Kluwer Academic publishers, Dordrecht, The Netherlands.

- Bentsen, R.G., (2005). Effect of neglecting interfacial coupling when using vertical flow experiments to determine relative permeability. Journal of Petroleum Science and Engineering, Volume 48, Issues 1-2, 30 July 2005, Pages 81-93.

- Bentsen, R.G., (2003). Interfacial coupling in vertical, two-phase flow through porous media. Paper Submitted to J. Petrol. Sci. Eng.

- Bentsen, R.G., (2001). The physical origin of interfacial coupling in two-phase flow through porous media. Transport in Porous Media. 44, 109-122.

- Bentsen, R.G., (1998a). Influence of hydrodynamic forces and interfacial momentum transfer on the flow of two immiscible phases. J. Petrol. Sci. Eng. 19, 177-190.

- Bentsen, R.G., (1998b). Effect of momentum transfer between fluid phases on effective mobility. J. Petrol. Sci. Eng. 21, 27-42.

- Bentsen, R.G., (1997). Impact of model error on the measurement of flow properties needed to describe flow through porous media. Revue de L'institut Francais du Petrole, Vol. 52(3), 299-315.

- Bentsen, R.G., (1994). Effect of hydrodynamic forces on capillary pressure and relative permeability. Transport in Porous Media. 17, 121-132.

- Bentsen, R.G., (1992). Construction and experimental testing of a new pressure-difference equation. AOSTRA J. Res. 8, 159-168.

- Bentsen, R.G., (1978). Conditions under which the capillary term can be neglected. JCPT. October - December 1978, 17(4), 25-30.

- Bentsen, R.G., (1976). Scaled fluid-flow models with permeabilities differing from that of the prototype. JCPT. July - September 1976, 15(3), 46-52.

- Bentsen, R.G. and Manai, A. A., (1993). On the use of conventional cocurrent and countercurrent effective permeabilities to estimate the four generalized permeability coefficients which arise in coupled, two-phase flow. Transport in Porous Media. 11, 243-262.

- Bentsen, R. G. and Manai A. A., (1991). Measurement of cocurrent and countercurrent relative permeability curves using the steady-state method. AOSTRA J. Res. 7, 169-181.

- Berg, C.R., (1995). A simple, effective-medium model for water saturation in porous rocks. Geophysics. 60(4), 1070-1080.

- Bona, N., Ortenzi, A. and Capaccioli, S., (2002). Advances in understanding the relationship between rock wettability and high-frequency dielectric response. J. Petrol. Sci. Eng. 33, 87-99.

- Bona, N., Rossi, E. and Capaccioli, S., (2001). Electrical measurements in the 100 Hz to 10 GHz frequency range for efficient rock wettability determination. SPE Reservoir Engineering. March 2001, 80-88.

- Bourbiaux, B.J. and Kalaydjian, F.J., (1990). Experimental study of cocurrent and countercurrent flows in natural porous media. SPE Reservoir Engineering Journal. August 1990, 361-368.

- Buckley, S.E. and Leverett, M.C., (1942). Mechanism of fluid displacements in sands. Trans. AIME, 146, 107-116.

- Cloud, W.F., (1930). Variation of pressure gradient with distance of rectilinear flow of gas-saturated oil and unsaturated oil through unconsolidated sands. Trans. AIME, Vol. 86, 337-350.

- Davis, Jr., L.A., (1980). VHF electrical measurement of saturation in laboratory floods. SPE Paper number 8847 presented at the 1980 1st joint SPE/DOE symposium on enhanced oil recovery, Tulsa, Oklahoma, April 20-23, 1980.

- de la Cruz and Spanos, T.J.T., (1983). Mobilization of oil ganglia. AIChE J. 29(5), 854-858.

- Fletcher, J.E., (1949). Some properties of water solutions that influence infiltration. Trans. Amer. Geophysical Union. Vol. 30(4), pp. 548-554.

- Hammervold, W.L. and Skjaeveland, S.M., (1992). Improvement of diaphragm method for drainage capillary pressure measurement with micro pore membrane. paper presented at the EUROCAS meeting, Sept. 8-10, Paris, France.

- Hammervold, W.L., Knutsen, O., Iversen, J.E. and Skjaeveland, S.M., (1998). Capillary pressure scanning curves by the micropore membrane technique. J. Petrol. Sci. Eng., Vol. 20, 253-258.

- Jones-Parra, J., Stahl, C.D., and Calhoun, J.C., Jr., (1954). A theoretical and experimental study of constant rate displacements in water-wet systems. Prod. Mon., 18-26, 1954.

- Kalaydjian, F., (1990). Origin and quantification of coupling between relative permeabilities for two-phase flows in porous media. Transport in Porous Media 5 (3), 215-229.

- Kalaydjian, F., (1987). A macroscopic description of multiphase flow in porous media involving space-time evolution of fluid/fluid interface. Transport in Porous Media. 2, 537-552.

- Kraszewski, A., (1996). Microwave aquametry - Electromagnetic wave interaction with water-containing materials. IEEE Press New York.

- Kyte, J.R. and Rapoport, L.A., (1958). Linear Waterflood behavior and end effects in water-wet porous media. Trans. AIME, Vol. 213, 423-426.

- Lelièvre, R.F., (1966). Etude d' ècoulements disphasiques permanents à contre-courants en milieu poreux - Comparaison avec les ècoulements de même sens (in French). Ph.D. Thesis, University of Toulouse, France, 1966.

- Leverett, M.C., (1941). Capillary behavior in porous solids. Trans. of AIME. 142,152-169.

- Liang, Q., (1993). Interaction Between Immiscible Phases Flowing at Different Velocities in Porous Media. D.E. Thesis, Louisiana Tech University, USA.

- Liang, Q. and Lohrenz, J., (1994). Dynamic method of measuring coupling coefficients of transport equations of two-phase flow in porous media. Transport in Porous Media. 15, 7179.

- Longeron, D., Hammervold, W.L. and Skjaeveland, S.M., (1994). Water-oil capillary pressure and wettability measurement using micropore technique. Paper presented at The International Symposium of the Society of Core Analysts, Sept. 12-14, Stavanger, Norway.

- Marle, C., (1981). Multiphase flow in porous media. Paris: Editions Technip, 1981.

- Muccino, J. Gray. W. and Ferrand, L., (1998). Towards an improved understanding of multiphase flow in porous media, Rev. Geophy. 36(3), 401-422.

- Muskat, M. and Meres, M.W., (1936). The flow of heterogeneous fluids through porous media. Physics. 7, 346-363.

- Muskat, M., Wyckoff, R.D., Botset, H.G. and Meres, M.W., (1937). Flow of gas-liquid mixtures through sands. Trans. AIME. 123, 69-96.

- Nasr, T.N., Law, D.H.S., Golbeck, H. and Korpany, G., (2000). Counter-current aspect of the SAGD process. JCPT. January 2000, 39(1), 41-47.

- Nguyen, B. -L., Geels, A.M., Bruining, J. and Slob, E.C., (1999). Calibration measurements of dielectric properties of porous media. SPE Journal. December 1999, 4(4), 353 -359.

- Onsager, L., (1931a). Reciprocal relations in irreversible processes-I. Physical Review. 37, 405-426.

- Onsager, L., (1931b). Reciprocal relations in irreversible processes-II. Physical Review. 38, 2265-2279.

- Orlov, S.I., (1970). Calculation and designing of coaxial resonators (in Russian). The Soviet Wireless, Moscow, USSR.

- Peaceman, D.P., (1977). Fundamentals of numerical reservoir simulation: Developments in petroleum engineering, 6). Elsevier Scientific Publishing Company, Amsterdam, The Netherlands, 65-82.

- Plummer, F.B., Hunter, J.C., Jr. and Timmerman, E.H., (1937). Flow of mixtures of oil and water through sand. API Drill. Prod. Pract., 417-421.

- Rakotomalala, N., Salin, D. and Yortsos, C.Y., (1995). Viscous coupling in a model porous medium geometry: Effect of fluid contact area. App. Scientific Res. 55, 155-169.

- Rose, W., (2000). Myths about later-day extension of Darcy's law. J. Petrol. Sci. Eng. 26, 187-198.

- Rose, W., (1999). Relative permeability ideas - Then and now (Richards to Leverett to Yuster, and Beyond). SPE paper number 57442 presented at the 1999 SPE regional meeting, Charleston, West Virginia, Oct. 1999.

- Rose, W., (1990). Lagrangian simulation of coupled two-phase flows. Mathematical Geology, Vol. 22(6), 641-654.

- Rose, W., (1988). Measuring transport coefficients necessary for the description of coupled two-phase flow of immiscible fluids in porous media. Transport in Porous Media, 3(2), 163-171.

- Sarma, H.K. and Bentsen, R.G., (1989). A new method for estimating relative permeabilities from unstabilized displacement data. JCPT. July-August 1989, 28(4), 118-128.

- Shen, C. and Ruth, D.W., (1996). Impact of inlet boundary conditions on the numerical simulation of one-dimensional coreflooding. JCPT. January 1996, 35(1), 19-24.

- Shen, C. and Ruth, D.W., (1994). Solutions to Bentsen's equation with finite-element method. J. Petrol. Sci. Eng. 11, 165-179.

- West, L.J., Handley, K., and Huang, Y., (2003). Radar frequency dielectric dispersion in sandstone: Implications for determination of moisture and clay content. Water Resources Research. February 2003. 39(2), 1:1-12.

- Whitaker, S., (1986). Flow in porous media 11: The governing equations for immiscible, two-phase flow. Transport in Porous Media. 1, 105-125.

- Yuster, S.T., (1951). Theoretical consideration of multiphase flow in idealized capillary systems. Proceeding of the Third World Petroleum Congress. 2, 437-445.

- Zarcone, C. and Lenormand, R., (1994). Dètermination expèrimentale du couplage visqueux dans les écoulements diphasiques en milieu poreux, C. R. Acad. Sci., Paris, Series II, 318, 1429-1438.

Appendix A: Coefficients for saturation calibration equations

Distance (cm)	*a*	*b*	*c*	*d*
66.0	45621.3446	-3864945.7666	-3.34E+08	2.83E+10
64.5	45621.3403	-3864945.7666	-3.34E+08	2.83E+10
63.1	45621.3246	-3864945.7668	-3.34E+08	2.83E+10
61.6	45621.3249	-3864945.7668	-3.34E+08	2.83E+10
60.1	45621.3364	-3864945.7667	-3.34E+08	2.83E+10
58.6	45621.3478	-3864945.7666	-3.34E+08	2.83E+10
57.2	45621.3395	-3864945.7667	-3.34E+08	2.83E+10
55.7	45621.3337	-3864945.7667	-3.34E+08	2.83E+10
54.2	45621.3363	-3864945.7667	-3.34E+08	2.83E+10
52.7	45621.3466	-3864945.7666	-3.34E+08	2.83E+10
51.3	45621.3494	-3864945.7665	-3.34E+08	2.83E+10
49.8	45621.3408	-3864945.7666	-3.34E+08	2.83E+10
48.3	45621.3338	-3864945.7667	-3.34E+08	2.83E+10
46.8	45621.3361	-3864945.7667	-3.34E+08	2.83E+10
45.4	45621.3376	-3864945.7667	-3.34E+08	2.83E+10
43.9	45621.3396	-3864945.7667	-3.34E+08	2.83E+10
42.4	45621.3410	-3864945.7666	-3.34E+08	2.83E+10
40.9	45621.3373	-3864945.7667	-3.34E+08	2.83E+10
39.5	45621.3304	-3864945.7668	-3.34E+08	2.83E+10
38.0	45621.3179	-3864945.7669	-3.34E+08	2.83E+10
36.5	45621.3251	-3864945.7668	-3.34E+08	2.83E+10
35.1	45621.3281	-3864945.7668	-3.34E+08	2.83E+10
33.6	45621.3325	-3864945.7667	-3.34E+08	2.83E+10
32.1	45621.3253	-3864945.7668	-3.34E+08	2.83E+10
30.6	45621.3238	-3864945.7668	-3.34E+08	2.83E+10
29.2	45621.3139	-3864945.7670	-3.34E+08	2.83E+10
27.7	45621.3119	-3864945.7670	-3.34E+08	2.83E+10
26.2	45621.3202	-3864945.7669	-3.34E+08	2.83E+10
24.7	45621.3284	-3864945.7668	-3.34E+08	2.83E+10
23.3	45621.3251	-3864945.7668	-3.34E+08	2.83E+10
21.8	45621.3216	-3864945.7669	-3.34E+08	2.83E+10
20.3	45621.3208	-3864945.7669	-3.34E+08	2.83E+10
18.8	45621.3206	-3864945.7669	-3.34E+08	2.83E+10
17.4	45621.3145	-3864945.7669	-3.34E+08	2.83E+10
15.9	45621.3224	-3864945.7669	-3.34E+08	2.83E+10
14.4	45621.3422	-3864945.7666	-3.34E+08	2.83E+10
12.9	45621.3563	-3864945.7665	-3.34E+08	2.83E+10
11.5	45621.3653	-3864945.7664	-3.34E+08	2.83E+10
10.0	45621.3778	-3864945.7662	-3.34E+08	2.83E+10

Appendix B: Phase pressure data of SSCO flow experiments

S_1	0.10		0.20		0.28		0.40	
Distance (m)	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2
0.1	106381.5	110557.2	123281.7	126892.5	147993.0	151273.0	144441.2	147331.2
0.2	**	97379.4	111848.1	113498.0	134177.5	136350.7	131639.1	133342.7
0.3	**	86369.6	99731.3	103490.3	120158.4	125137.1	117223.1	121266.6
0.4	**	***	82999.8	***	101936.9	***	98461.2	***
0.5	**	55924.3	59303.9	67285.2	82000.0	84482.8	72100.0	80056.1
0.6	**	42247.8	48392.2	51235.6	64644.6	64805.9	54236.4	62118.6
0.7	**	28698.3	34755.8	37013.1	44287.8	46837.5	37903.8	44411.1

Continued

S_1	0.47		0.57		0.73		0.88	
Distance (m)	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2
0.1	96697.2	99347.2	79692.4	82072.4	65328.5	67593.4	12437.2	14237.2
0.2	88328.7	89701.5	71512.4	72999.6	57860.8	58520.6	10774.9	**
0.3	79963.3	83090.1	64148.8	67346.6	50497.2	52867.6	9598.0	**
0.4	67732.7	***	53667.4	***	40015.8	***	8350.0	**
0.5	54728.5	56629.4	41996.7	44642.2	28345.0	30163.2	6287.1	**
0.6	41273.4	44891.1	34626.2	34802.0	19323.2	20323.0	4835.1	**
0.7	29946.0	32811.5	23291.2	25561.6	9639.6	11082.6	3337.5	**

, did not measure; *, malfunction of pressure transducer

Appendix C: Relative permeability data

S_1	Water phase			Oil phase		
	q_1	dP$_1$/dx	k_1	q_2	dP$_2$dx	k_2
	(m^3/s)	(Pa/m)	(m^2)	(m^3/s)	(Pa/m)	(m^2)
0.10	0	**	0.000E+00	1.583E-08	137959	8.721E-12
0.20	3.170E-09	154613	4.100E-14	1.531E-08	153703	7.568E-12
0.28	6.633E-09	174407	7.607E-14	1.555E-08	177518	6.656E-12
0.40	1.326E-08	185550	1.430E-13	1.226E-08	175864	5.300E-12
0.47	1.153E-08	114143	2.020E-13	6.701E-09	112746	4.517E-12
0.57	1.576E-08	94689	3.329E-13	4.298E-09	95940	3.405E-12
0.73	3.533E-08	95105	7.430E-13	2.073E-09	95940	1.642E-12
0.88	1.147E-08	15175	1.511E-12	0	**	0

**, not available

Appendix D: Source code of the simulator

1) The source codes listings for Ics.java

```
package ics;

import java.awt.*;
import ics.*;

/**
 * <p>Title: Interfacial Coupling Simulator (2.0)</p>
 * <p>Description: Program for Testing Coupling effects</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>University: University of Alberta</p>
 * <p>Program: Petroleum Engineering</p>
 * <p>Degree: Master of Science - M.Sc</p>
 * @author: Xiao Y Zhang
 * @version 2.0
 * @version 1.0 author: O.R. Ayodele
 */

//This class is the main class of the simulator and provides the entrance to the package.


public class Ics {

  boolean packFrame = false;

  //Construct the application
  public Ics() {
    IcsFrame frame = new IcsFrame();

    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
     frame.pack();
    }
    else {
     frame.validate();
    }

    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();

    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }

    if (frameSize.width > screenSize.width) {
     frameSize.width = screenSize.width;
    }

    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
 }

 //Main method
 public static void main(String[] args) {
   new Ics();
  }
}
```

2) The source codes listings for IcsFrame.java

```
package ics;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import ics.*;

//This class is to construct a main frame which will contain other GUI Components.

public class IcsFrame extends JFrame {
 //Panel
 JPanel contentPane;

 //Menu bar and menu items
 JMenuBar jMenuBar1 = new JMenuBar();
 JMenu jMenuFile = new JMenu();
 JMenuItem jMenuFileNew = new JMenuItem();
 JMenuItem jMenuFileExit = new JMenuItem();
 JMenu jMenuHelp = new JMenu();
 JMenuItem jMenuHelpAbout = new JMenuItem();

 //Buttons
 JButton jButton1 = new JButton();
 JButton jButton2 = new JButton();
 JButton jButton3 = new JButton();

 //Image Icons
 ImageIcon image1;
 ImageIcon image2;
 ImageIcon image3;
 JMenuItem jMenuNew = new JMenuItem();

 //Border
 Border border1;

 //Grid layout
 GridLayout gridLayout1 = new GridLayout();

  //Construct the frame
 public IcsFrame() {
   try {
    jbInit();
   }

   catch(Exception e) {
     e.printStackTrace();
   }
 }

 //Component initialization
 private void jbInit() throws Exception  {
   contentPane = (JPanel) this.getContentPane();
  border1 = BorderFactory.createEtchedBorder(Color.white,new Color(134,134,134));
  contentPane.setLayout(gridLayout1);
  contentPane.setBackground(new Color(58, 110, 165));
  this.setSize(new Dimension(870, 700));
  this.setTitle("Interfacial Coupling Simulator(2.0)");
  jMenuFile.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuFile.setMnemonic('F');
  jMenuFile.setText("File");
  jMenuFileNew.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuFileNew.setMnemonic('N');
  jMenuFileNew.setText("New");
  jMenuFileNew.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuFileNew_actionPerformed(e);
    }
  });
  jMenuFileExit.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuFileExit.setMnemonic('E');
  jMenuFileExit.setText("Exit");
  jMenuFileExit.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuFileExit_actionPerformed(e);
    }
  });

  jMenuHelp.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuHelp.setMnemonic('H');
  jMenuHelp.setText("Help");
  jMenuHelpAbout.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuHelpAbout.setMnemonic('A');
  jMenuHelpAbout.setText("About");

  jMenuHelpAbout.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuHelpAbout_actionPerformed(e);
    }
  });

  jMenuNew.setFont(new java.awt.Font("Dialog", 1, 12));
  jMenuNew.setMnemonic('N');
  jButton1.setBorder(border1);
  contentPane.setEnabled(true);
  jMenuFile.add(jMenuFileNew);
  jMenuFile.addSeparator();
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuHelp);
  this.setJMenuBar(jMenuBar1);
  this.toBack();
}

 //File | New action performed
 public void jMenuFileNew_actionPerformed(ActionEvent e) {
   JDesktopPane n = new JDesktopPane();
   n.setBackground(new Color(58, 110, 165));
   this.getContentPane().add(n);

   //An instance of IcsClass
   IcsClass inputbox = new IcsClass();
```

```
    //Set inputbox size and display
    Dimension dlgSize = getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    inputbox.jif.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 +
loc.y);
    inputbox.jif.setResizable(false);
    inputbox.jif.setBounds(40, 150, 825, 430);
    inputbox.jif.show();
    n.add(inputbox.jif);
    inputbox.jif.setLocation(20, 213);
    inputbox.jif.setBorder(BorderFactory.createEtchedBorder(Color.white,new Color(134,134,134)));
    inputbox.jif.setTitle("Simulation Setting");
    inputbox.f1.setLocation(20, 1);
    inputbox.f1.setResizable(false);
    inputbox.f1.setSize(inputbox.fv1.width, inputbox.fv1.height);
    inputbox.f1.setVisible(true);
    inputbox.f1.setBorder(BorderFactory.createEtchedBorder(Color.white,new Color(134,134,134)));

    inputbox.f2.setLocation(250, 1);
    inputbox.f2.setResizable(false);
    inputbox.f2.setSize(inputbox.fv2.width, inputbox.fv2.height);
    inputbox.f2.setVisible(true);
    inputbox.f2.setBorder(BorderFactory.createEtchedBorder(Color.white,new Color(134,134,134)));

    n.add(inputbox.f2);
    n.add(inputbox.f1);

    inputbox.ControlOption.setLocation(615, 1);
    inputbox.ControlOption.setResizable(false);
    inputbox.ControlOption.setSize(inputbox.fv2.width+20, inputbox.fv2.height);
    inputbox.ControlOption.setVisible(true);
    inputbox.ControlOption.setBorder(BorderFactory.createEtchedBorder(Color.white,new Color(134,134,134)));

    n.add(inputbox.ControlOption);

    //Progress bar code for monitoring simuation run status
    inputbox.jProgressBar1.setIndeterminate(false);

    //Reset all of these values to zeros and initial title
    inputbox.flowCount = 0;
    inputbox.temp_inS1=0;
    inputbox.MaxAbsMatrixSolution = 0;
    inputbox.timeDivision = 0;
    inputbox.flowIterationNumber = 0;
    for(int i = 0; i < inputbox.bNum; i++){
      inputbox.copyE[i] = 0;
    }

  }

  //File | Exit action performed
  public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }

  //Help | About action performed
  public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
    IcsFrame_AboutBox dlg = new IcsFrame_AboutBox(this);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
  }

  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);

    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      jMenuFileExit_actionPerformed(null);
    }
  }
}
```

3) The source codes listings for IcsClass.java

```
package ics;

import java.awt.*;
import java.awt.event.*;
import java.awt.FileDialog.*;
import java.io.*;
import java.io.Reader.*;
import java.text.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import ics.*;

/** IcsClass (interfacial coupling class) for solving Bentsen's Equation
 * which incorporates interfacial coupling using the fully-implicit finite
 * difference method and considering the inlet end effect.The solution of
 * the resulting systems of linear equations is based on the "ThomasAlgorithmSolution"
 * and "newtonJacobi" methods, which are objects in the IcsClass class.
 * @author: Xiao Y Zhang
 * @version 2.0
 * @version 1.0 author: O.R. Ayodele
 */

public class IcsClass {

  /**
   * Initiation of Components (Constructor)
   */
  public IcsClass() {
    try {
      initiate();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    try {
      jbInit();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
}
//CONSTRUCTION OF GUI STARTS HERE

 //Declaration of all public and common GUI variables (fields) starts here
 JInternalFrame jif = new JInternalFrame();
 JInternalFrame ControlOption = new JInternalFrame("Control Options");
 JProgressBar jProgressBar1 = new JProgressBar();
 DecimalFormat dataFormatUpdateTime2 = new DecimalFormat("0.00");
 DecimalFormat dataFormatUpdateTime8 = new DecimalFormat("0.00000000");
 DecimalFormat dataFormat6 = new DecimalFormat("0.000000");
 JInternalFrame f1 = new JInternalFrame("Fractional Flow Profile");
 JInternalFrame f2 = new JInternalFrame("Saturation Profile");
 FlowView fv1 = new FlowView();
 FlowView fv2 = new FlowView();
 private JPanel jPanel1 = new JPanel();
 private JPanel jPanel2 = new JPanel();
 private JPanel jPanel3 = new JPanel();
 private JPanel jPanel4 = new JPanel();
 private Border border1;
 private TitledBorder titledBorder1;
 private Border border2;
 private TitledBorder titledBorder2;
 private Border border3;
 private TitledBorder titledBorder3;
 private Border border4;
 private TitledBorder titledBorder4;
 private JButton runSimulation = new JButton();
 private Border border5;
 private TitledBorder titledBorder5;
 private JTextField jTextField1 = new JTextField();
 private JTextField jTextField2 = new JTextField();
 private JTextField jTextField3 = new JTextField();
 private JLabel jLabel1 = new JLabel();
 private JTextField jTextField4 = new JTextField();
 private JLabel jLabel2 = new JLabel();
 private JLabel jLabel3 = new JLabel();
 private JLabel jLabel4 = new JLabel();
 private JLabel jLabel5 = new JLabel();
 private JLabel jLabel6 = new JLabel();
 private JLabel jLabel7 = new JLabel();
 private JTextField jTextField5 = new JTextField();
 private JTextField jTextField6 = new JTextField();
 private JTextField jTextField7 = new JTextField();
 private JTextField jTextField13 = new JTextField();
 private JTextField jTextField12 = new JTextField();
 private JTextField jTextField11 = new JTextField();
 private JTextField jTextField10 = new JTextField();
 private JTextField jTextField9 = new JTextField();
 private JTextField jTextField8 = new JTextField();
 private JLabel jLabel8 = new JLabel();
 private JLabel jLabel9 = new JLabel();
 private JLabel jLabel10 = new JLabel();
 private JLabel jLabel11 = new JLabel();
 private JLabel jLabel12 = new JLabel();
 private JLabel jLabel13 = new JLabel();
 private JLabel jLabel14 = new JLabel();
 private JTextField jTextField14 = new JTextField();
 private JTextField jTextField15 = new JTextField();
 private JTextField jTextField16 = new JTextField();
private JLabel jLabel15 = new JLabel();
private JTextField jTextField17 = new JTextField();
private JLabel jLabel16 = new JLabel();
private JTextField jTextField18 = new JTextField();
private JLabel jLabel17 = new JLabel();
private JTextField jTextField19 = new JTextField();
private JLabel jLabel18 = new JLabel();
private JLabel jLabel19 = new JLabel();
private JLabel jLabel20 = new JLabel();
private JLabel jLabel21 = new JLabel();
private JTextField jTextField20 = new JTextField();
private JTextField jTextField21 = new JTextField();
private JButton jButton5 = new JButton();
private JButton stopSimulation = new JButton();
private JPanel jPanel6 = new JPanel();
private JPanel jPanel7 = new JPanel();
private JTextField jTextField22 = new JTextField();
private JLabel jLabel22 = new JLabel();
private JTextField jTextField23 = new JTextField();
private JLabel jLabel23 = new JLabel();
private TitledBorder titledBorder6;
private TitledBorder titledBorder7;
private JSlider jSlider1 = new JSlider();
private JPanel jPanel9 = new JPanel();
private Border border6;
private Border border7;
private TitledBorder titledBorder8;
private JPanel jPanel8 = new JPanel();
private Border border8;
private TitledBorder titledBorder9;
private TitledBorder titledBorder92;
private JButton clearAll = new JButton();
private Border border9;
private TitledBorder titledBorder10;
private ButtonGroup R12 = new ButtonGroup();
private ButtonGroup Flow = new ButtonGroup();
private ButtonGroup Coupling = new ButtonGroup();
private ButtonGroup SolutionMethod = new ButtonGroup();
private ButtonGroup Solver = new ButtonGroup();
private JRadioButton r12 = new JRadioButton();
private JRadioButton jRadioButton2 = new JRadioButton();
private Border border10;
private TitledBorder titledBorder11;
private Border border11;
private Border border12;
private TitledBorder titledBorder12;
private JTextField jTextField24 = new JTextField();
private JTextField jTextField25 = new JTextField();
private JLabel jLabel24 = new JLabel();
private JLabel jLabel25 = new JLabel();
private Border border13;
private TitledBorder titledBorder13;
private JRadioButton viscousCoupling = new JRadioButton();
private JRadioButton capillaryCoupling = new JRadioButton();
private JRadioButton bothCouplings = new JRadioButton();
private JRadioButton noCoupling = new JRadioButton();
private JPanel jPanel5 = new JPanel();
private Border border14;
private TitledBorder titledBorder14;
private JTextField jTextField27 = new JTextField();
private JTextField jTextField28 = new JTextField();
private JRadioButton Fixed = new JRadioButton();
private JRadioButton Changeable = new JRadioButton();
private ButtonGroup InletSaturation = new ButtonGroup();
private Border border15;
private TitledBorder titledBorder15;
private JLabel jLabel29 = new JLabel();
private JLabel jLabel210 = new JLabel();
private Border border16;
private TitledBorder titledBorder16;
private JPanel jPanel10 = new JPanel();
private Border border17;
private TitledBorder titledBorder17;
private JTextField jTextField30 = new JTextField();
private JTextField jTextField31 = new JTextField();
private JTextField jTextField32 = new JTextField();
private JTextField jTextField33 = new JTextField();
private JLabel jLabel27 = new JLabel();
private JLabel jLabel211 = new JLabel();
private JLabel jLabel212 = new JLabel();
private JLabel jLabel213 = new JLabel();
private Border border18;
private JLabel jLabel214 = new JLabel();
private JTextField jTextField37 = new JTextField();
private JLabel jLabel215 = new JLabel();
private JLabel jLabel216 = new JLabel();
private JTextField jTextField36 = new JTextField();
private JTextField jTextField35 = new JTextField();
private JTextField jTextField34 = new JTextField();
private JLabel jLabel217 = new JLabel();
private JPanel jPanel13 = new JPanel();
private Border border19;
private TitledBorder titledBorder18;
private Border border20;
private Border border21;
private TitledBorder titledBorder19;
private JTextField jTextField41 = new JTextField();
private JLabel jLabel219 = new JLabel();
private JTextField jTextField40 = new JTextField();
private JTextField jTextField39 = new JTextField();
private JTextField jTextField38 = new JTextField();
private Border border22;
private TitledBorder titledBorder20;
private JTextField jTextField29 = new JTextField();
private Border border23;
private TitledBorder titledBorder21;
private JLabel jLabel2112 = new JLabel();
private Border border24;
private TitledBorder titledBorder22;
private JButton saveInputData = new JButton();
private JButton openInputData = new JButton();
private Border border25;
private TitledBorder titledBorder23;
private Border border26;
private TitledBorder titledBorder24;
private Border border27;
private JLabel jLabel220 = new JLabel();
private Border border28;
private JLabel jLabel2110 = new JLabel();
private JLabel jLabel2111 = new JLabel();
private JLabel jLabel2113 = new JLabel();
private JPanel jPanel14 = new JPanel();
private JPanel jPanel11 = new JPanel();
private Border border29;
private TitledBorder titledBorder25;
private Border border30;
private TitledBorder titledBorder26;
private Border border31;
private TitledBorder titledBorder27;
private Border border32;
private JRadioButton implicitNewtonRaphson = new JRadioButton();
private JRadioButton implicitNewtonJacobi = new JRadioButton();
private JRadioButton no = new JRadioButton();
private JRadioButton yes = new JRadioButton();
private Border border33;
private TitledBorder titledBorder28;
private JPanel jPanel15 = new JPanel();
private Border border34;
private TitledBorder titledBorder29;
private Border border35;
private TitledBorder titledBorder30;
private Border border36;
private TitledBorder titledBorder31;
private JPanel jPanel16 = new JPanel();
private Border border37;
private JScrollPane jScrollPane1 = new JScrollPane();
boolean isStop = true;

//Declaration of all public and common GUI variables (fields) ends here*/

//Initiation of GUI Starts here
private void initiate() throws Exception {
  border1 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder1 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
     white, new Color(134, 134, 134)), "Fluid Properties");
  border2 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder2 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
     white, new Color(134, 134, 134)), "Reservoir Properties");
  border3 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder3 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
     white, new Color(134, 134, 134)), "Simulation Settings");
  border4 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder4 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
     white, new Color(134, 134, 134)), "Saturation Block (Grid)");
  titledBorder6 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
     white, new Color(134, 134, 134)), "Hydrodynamic Effect (R12)");
  border5 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder5 = new TitledBorder(BorderFactory.createEtchedBorder(Color. white, new Color(134, 134, 134)),
"");
  border6 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  border7 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder8 = new TitledBorder(new EtchedBorder(EtchedBorder.RAISED,
     Color.white, new Color(134, 134, 134)), "Coupling Options");
  border8 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder9 = new TitledBorder(BorderFactory.createEtchedBorder(Color. white, new Color(134, 134, 134)),
"Pc Coeff");
  border9 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder10 = new TitledBorder(border9, "Events");
  border10 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder11 = new TitledBorder(border10, "Events");
  border11 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134)),
"Event");
  border12 = new EtchedBorder(EtchedBorder.RAISED, Color.white,  new Color(134, 134, 134));
  titledBorder12 = new TitledBorder(border12, "Events");
  border13 = new EtchedBorder(EtchedBorder.RAISED, Color.white,  new Color(134, 134, 134));
  border14 = new EtchedBorder(EtchedBorder.RAISED, Color.white,  new Color(134, 134, 134));
  titledBorder14 = new TitledBorder(BorderFactory.createEmptyBorder(), "Event");
  border15 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder15 = new TitledBorder(border15, "Viscous Coupling Term");
  border16 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  border17 = new EtchedBorder(EtchedBorder.RAISED, Color.white,  new Color(134, 134, 134));
  titledBorder17 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
    white, new Color(134, 134, 134)), "Inlet Saturation (S*)");
  border18 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134)), "Kro
Coeff");
  border19 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder18 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
    white, new Color(134, 134, 134)), "Krw Coeff");
  border20 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white,
    new Color(134, 134, 134)), "Krw Coeff");
  border21 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder19 = new TitledBorder(border21, "Kro Coeff");
  border22 = BorderFactory.createEmptyBorder();
  titledBorder20 = new TitledBorder(new EtchedBorder(EtchedBorder.RAISED,
    Color.white, new Color(134, 134, 134)), "Kro Coeff");
  border23 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder21 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
    white, new Color(134, 134, 134)), "Events");
  border24 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder22 = new TitledBorder(border24, "Viscous Coupling Term");
  border25 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder23 = new TitledBorder(border25, "Events");
  border26 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder24 = new TitledBorder(border26, "Events");
  border27 = new EtchedBorder(EtchedBorder.RAISED, new Color(197, 187, 178), new Color(96, 91, 87));
  border28 = BorderFactory.createEmptyBorder();
  border29 = BorderFactory.createEmptyBorder();
  titledBorder25 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
    white, new Color(134, 134, 134)), "Initial Conditions");
  border30 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  titledBorder26 = new TitledBorder(BorderFactory.createEtchedBorder(Color.
    white, new Color(134, 134, 134)), "Coupling");
  border31 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  border32 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134)),
"Interactive");
  border33 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  titledBorder28 = new TitledBorder(BorderFactory.createEtchedBorder(Color. white, new Color(134, 134, 134)),
"Solver ");
  border34 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  border35 = BorderFactory.createEtchedBorder(Color.white, new Color(134, 134, 134));
  border36 = new EtchedBorder(EtchedBorder.RAISED, Color.white, new Color(134, 134, 134));
  border37 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white,
    new Color(134, 134, 134)), "Viscous Coupling ");
  jif.getContentPane().setLayout(null);
  jPanel1.setBorder(titledBorder1);
  jPanel1.setBounds(new Rectangle(5, 2, 372, 236));
  jPanel1.setLayout(null);
  jPanel2.setLayout(null);
  jPanel2.setBounds(new Rectangle(379, 3, 221, 236));
  jPanel2.setBorder(titledBorder2);
  jPanel3.setLayout(null);
  jPanel3.setBounds(new Rectangle(4, 233, 202, 163));
  jPanel3.setBorder(titledBorder3);
  jPanel4.setLayout(null);
```

```
jPanel4.setBorder(titledBorder25);
jPanel4.setBounds(new Rectangle(207, 309, 166, 89));
jif.setFont(new java.awt.Font("Dialog", 1, 12));
jif.setResizable(true);
jif.setTitle("Simulation Setting");
runSimulation.setText("Run Simulation");
runSimulation.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    runSimulation_actionPerformed(e);
  }
});
runSimulation.setBorder(BorderFactory.createEtchedBorder());
runSimulation.setMnemonic('R');
runSimulation.setBounds(new Rectangle(662, 340, 124, 17));
jTextField1.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField1.setText("812.3");
jTextField1.setBounds(new Rectangle(127, 14, 57, 24));
jTextField2.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField2.setText("998.2");
jTextField2.setBounds(new Rectangle(127, 44, 57, 24));
jTextField3.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField3.setText("0.04762");
jTextField3.setBounds(new Rectangle(127, 75, 57, 24));
jLabel1.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel1.setText("Oil Density (Kg/m^3)");
jLabel1.setBounds(new Rectangle(10, 16, 108, 29));
jTextField4.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField4.setText("0.0010");
jTextField4.setBounds(new Rectangle(127, 105, 57, 24));
jLabel2.setBounds(new Rectangle(10, 47, 114, 29));
jLabel2.setText("Water Density (Kg/m^3)");
jLabel2.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel3.setBounds(new Rectangle(10, 77, 108, 29));
jLabel3.setText("Oil Viscosity (Pa.s)");
jLabel3.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel4.setBounds(new Rectangle(10, 108, 108, 29));
jLabel4.setText("Water Viscosity (Pa.s)");
jLabel4.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel5.setBounds(new Rectangle(10, 139, 108, 29));
jLabel5.setText("Injection Time (sec)");
jLabel5.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel6.setBounds(new Rectangle(10, 169, 112, 29));
jLabel6.setText("Water Inj. Rate (m^3/s)");
jLabel6.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel7.setBounds(new Rectangle(10, 200, 126, 29));
jLabel7.setText("Flow No. of Time Step");
jLabel7.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField5.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField5.setText("360");
jTextField5.setBounds(new Rectangle(127, 135, 57, 24));
jTextField6.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField6.setText("9.0E-9");
jTextField6.setBounds(new Rectangle(127, 166, 57, 24));
jTextField7.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField7.setText("5");
jTextField7.setBounds(new Rectangle(127, 196, 57, 24));
jTextField13.setBounds(new Rectangle(308, 167, 57, 24));
jTextField13.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField13.setText("0.00001");
jTextField12.setBounds(new Rectangle(308, 136, 57, 24));
jTextField12.setBorder(BorderFactory.createLineBorder(Color.black));
```

```
jTextField12.setText("50");
jTextField11.setBounds(new Rectangle(308, 106, 57, 24));
jTextField11.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField11.setText("0.2000");
jTextField10.setBounds(new Rectangle(308, 76, 57, 24));
jTextField10.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField10.setText("0.1600");
jTextField9.setBounds(new Rectangle(308, 45, 57, 24));
jTextField9.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField9.setText("9.9E-12");
jTextField8.setBounds(new Rectangle(308, 15, 57, 24));
jTextField8.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField8.setText("9.0E-12");
jLabel8.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel8.setText("Length (m)");
jLabel8.setBounds(new Rectangle(8, 16, 126, 29));
jLabel9.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel9.setText("Flow Covg. Tolerance");
jLabel9.setBounds(new Rectangle(191, 170, 112, 29));
jLabel10.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel10.setText("Flow Count Tolerance");
jLabel10.setBounds(new Rectangle(191, 140, 108, 29));
jLabel11.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel11.setText("Sor (fraction)");
jLabel11.setBounds(new Rectangle(191, 109, 108, 29));
jLabel12.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel12.setText("Swi (fraction)");
jLabel12.setBounds(new Rectangle(191, 78, 108, 29));
jLabel13.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel13.setText("Eff.K to Wat. (m^2)");
jLabel13.setBounds(new Rectangle(191, 48, 122, 29));
jLabel14.setBounds(new Rectangle(191, 17, 117, 29));
jLabel14.setText("Eff.K to Oil (m^2)");
jLabel14.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel14.setToolTipText("");
jTextField14.setBounds(new Rectangle(151, 21, 57, 16));
jTextField14.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField14.setText("0.60");
jTextField15.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField15.setText("0.10");
jTextField15.setBounds(new Rectangle(151, 47, 57, 16));
jTextField16.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField16.setText("0.09");
jTextField16.setBounds(new Rectangle(151, 72, 57, 16));
jLabel15.setToolTipText("");
jLabel15.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel15.setText("Thickness (m)");
jLabel15.setBounds(new Rectangle(10, 42, 117, 29));
jTextField17.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField17.setText("0.33");
jTextField17.setBounds(new Rectangle(151, 98, 57, 16));
jLabel16.setBounds(new Rectangle(10, 67, 117, 29));
jLabel16.setText("Height/Elevation (m)");
jLabel16.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField18.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField18.setText("0.0");
jTextField18.setBounds(new Rectangle(151, 123, 57, 16));
jLabel17.setBounds(new Rectangle(10, 196, 136, 29));
jLabel17.setText("Kro Fit Eq. (1 or 2)");
jLabel17.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel18.setBounds(new Rectangle(10, 93, 117, 29));
```

```
jLabel18.setText("Porosity (fraction)");
jLabel18.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel19.setBounds(new Rectangle(10, 119, 117, 29));
jLabel19.setText("Inclination (Degrees)");
jLabel19.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel20.setBounds(new Rectangle(10, 170, 145, 29));
jLabel20.setText("Krw Fit Eq. (1 or 2)");
jLabel20.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel21.setBounds(new Rectangle(10, 145, 117, 29));
jLabel21.setText("Ac,area of Pc curve(Pa)");
jLabel21.setFont(new java.awt.Font("Dialog", 1, 9));
jLabel21.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField19.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField19.setText("2500");
jTextField19.setBounds(new Rectangle(151, 149, 57, 16));
jTextField20.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField20.setText("2.0");
jTextField20.setBounds(new Rectangle(151, 174, 57, 16));
jTextField21.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField21.setToolTipText("");
jTextField21.setText("2.0");
jTextField21.setBounds(new Rectangle(151, 200, 57, 16));
jButton5.setBounds(new Rectangle(205, 197, 159, 24));
jButton5.setBorder(BorderFactory.createEtchedBorder());
jButton5.setMnemonic('C');
jButton5.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jButton5_actionPerformed(e);
  }
});
jButton5.setText("Clear Fluid Properties");
stopSimulation.setBounds(new Rectangle(662, 320, 124, 17));
stopSimulation.setBorder(BorderFactory.createEtchedBorder());
stopSimulation.setToolTipText("");
stopSimulation.setMnemonic('U');
stopSimulation.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    stopSimulation_actionPerformed(e);
  }
});
stopSimulation.setText("Stop Simulation");
jPanel7.setLayout(null);
jPanel7.setBounds(new Rectangle(4, 79, 187, 76));
jPanel7.setBorder(titledBorder4);
jTextField22.setBounds(new Rectangle(115, 37, 51, 18));
jTextField22.setText("0.00");
jTextField22.setEnabled(false);
jTextField22.setBorder(BorderFactory.createLineBorder(Color.black));
jLabel22.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel22.setText("Value of a for  R12");
jLabel22.setBounds(new Rectangle(17, 31, 94, 29));
jTextField23.setEnabled(false);
jTextField23.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField23.setText("10");
jTextField23.setBounds(new Rectangle(116, 53, 57, 16));
jTextField23.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    jTextField23_actionPerformed(e);
  }
});
jLabel23.setBounds(new Rectangle(16, 48, 69, 29));
```

```
jLabel23.setText("Block Number");
jLabel23.setFont(new java.awt.Font("Dialog", 1, 10));
jSlider1.setMajorTickSpacing(10);
jSlider1.setMinimum(10);
jSlider1.setMaximum(500);
jSlider1.setMinorTickSpacing(1);
jSlider1.setPaintTicks(true);
jSlider1.setValue(10);
jSlider1.setBorder(BorderFactory.createEtchedBorder());
jSlider1.setBounds(new Rectangle(15, 18, 159, 31));
jSlider1.addChangeListener(new ChangeListener() {
  public void stateChanged(ChangeEvent e) {
    jSlider1_actionChange(e);
  }
});
jPanel9.setBorder(border32);
jPanel9.setBounds(new Rectangle(207, 234, 80, 68));
jPanel9.setLayout(null);
no.setBounds(new Rectangle(10, 22, 65, 12));
no.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    no_actionPerformed(e);
  }
});
no.setText("No");
no.setMnemonic('S');
no.setBorder(null);
no.setFont(new java.awt.Font("Dialog", 1, 10));
yes.setBounds(new Rectangle(10, 45, 65, 12));
yes.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    yes_actionPerformed(e);
  }
});
yes.setText("Yes");
yes.setMnemonic('U');
yes.setSelected(true);
yes.setBorder(null);
yes.setFont(new java.awt.Font("Dialog", 1, 10));
jPanel8.setBorder(titledBorder9);
jPanel8.setBounds(new Rectangle(609, 128, 110, 113));
clearAll.setBounds(new Rectangle(662, 301, 124, 17));
clearAll.setBorder(BorderFactory.createEtchedBorder());
clearAll.setMnemonic('L');
clearAll.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    clearAll_actionPerformed(e);
  }
});
clearAll.setText("Clear All");
r12.setFont(new java.awt.Font("Dialog", 1, 10));
r12.setBorder(null);
r12.setMnemonic('C');
r12.setText("Compute R12");
r12.setBounds(new Rectangle(12, 21, 86, 13));
r12.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    r12_actionPerformed(e);
  }
});
jRadioButton2.setBounds(new Rectangle(113, 21, 61, 13));
```

```
jRadioButton2.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   jRadioButton2_actionPerformed(e);
  }
});
jRadioButton2.setText(" R12=1");
jRadioButton2.setMnemonic('R');
jRadioButton2.setSelected(true);
jRadioButton2.setBorder(null);
jRadioButton2.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField24.setBounds(new Rectangle(110, 24, 40, 20));
jTextField24.setText("1.0");
jTextField24.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField25.setBounds(new Rectangle(110, 56, 40, 20));
jTextField25.setText("1.0");
jTextField25.setBorder(BorderFactory.createLineBorder(Color.black));
jLabel24.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel24.setText("Assumed f(n+1)");
jLabel24.setBounds(new Rectangle(9, 19, 96, 29));
jLabel25.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel25.setText("f(S,t) When t=0");
jLabel25.setBounds(new Rectangle(9, 50, 96, 29));
viscousCoupling.setFont(new java.awt.Font("Dialog", 1, 10));
viscousCoupling.setBorder(null);
viscousCoupling.setMnemonic('0');
viscousCoupling.setSelected(true);
viscousCoupling.setText("Viscous");
viscousCoupling.setBounds(new Rectangle(743, 147, 65, 23));
viscousCoupling.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   viscousCoupling_actionPerformed(e);
  }
});
capillaryCoupling.setFont(new java.awt.Font("Dialog", 1, 10));
capillaryCoupling.setBorder(null);
capillaryCoupling.setMnemonic('0');
capillaryCoupling.setText("Capillary");
capillaryCoupling.setBounds(new Rectangle(743, 168, 65, 23));
capillaryCoupling.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   capillaryCoupling_actionPerformed(e);
  }
});
bothCouplings.setFont(new java.awt.Font("Dialog", 1, 10));
bothCouplings.setBorder(null);
bothCouplings.setActionCommand("bothCouplings");
bothCouplings.setMnemonic('C');
bothCouplings.setText("Both");
bothCouplings.setBounds(new Rectangle(743, 189, 65, 23));
bothCouplings.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   bothCouplings_actionPerformed(e);
  }
});
noCoupling.setFont(new java.awt.Font("Dialog", 1, 10));
noCoupling.setBorder(null);
noCoupling.setMnemonic('C');
noCoupling.setText("None");
noCoupling.setBounds(new Rectangle(743, 210, 65, 23));
noCoupling.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    noCoupling_actionPerformed(e);
  }
});
jPanel5.setBorder(titledBorder26);
jPanel5.setBounds(new Rectangle(723, 127, 94, 113));
Fixed.setFont(new java.awt.Font("Dialog", 1, 10));
Fixed.setBorder(null);
Fixed.setSelected(false);
Fixed.setMnemonic('0');
Fixed.setText("Fixed");
Fixed.setVisible(true);
Fixed.setBounds(new Rectangle(401, 266, 63, 20));
Fixed.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    Fixed_actionPerformed(e);
  }
});

Changeable.setFont(new java.awt.Font("Dialog", 1, 10));
Changeable.setBorder(null);
Changeable.setSelected(true);
Changeable.setMnemonic('0');
Changeable.setText("Changeable");
Changeable.setBounds(new Rectangle(484, 266, 103, 20));
Changeable.setVisible(true);
Changeable.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    Changeable_actionPerformed(e);
  }
});

jTextField27.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField27.setText("1.00");
jTextField27.setBounds(new Rectangle(521, 290, 97, 20));
jTextField28.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField28.setText("0.30");
jTextField28.setBounds(new Rectangle(521, 313, 97, 20));
jLabel29.setBounds(new Rectangle(401, 285, 120, 29));
jLabel29.setText("MBE CheckRatio (%)");
jLabel29.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel210.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel210.setText("Initial Assumed S*");
jLabel210.setBounds(new Rectangle(401, 312, 124, 29));
jPanel10.setBorder(titledBorder17);
jPanel10.setBounds(new Rectangle(382, 241, 246, 100));
jTextField30.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField30.setText("0.00");
jTextField30.setBounds(new Rectangle(644, 146, 56, 19));
jTextField31.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField31.setText("1.00");
jTextField31.setBounds(new Rectangle(644, 167, 56, 19));
jTextField32.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField32.setText("10.01");
jTextField32.setBounds(new Rectangle(644, 189, 56, 19));
jTextField33.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField33.setText("-9.98");
jTextField33.setBounds(new Rectangle(644, 210, 56, 19));
jLabel27.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel27.setText("a0");
jLabel27.setBounds(new Rectangle(616, 150, 12, 15));
jLabel211.setFont(new java.awt.Font("Dialog", 1, 10));
```

```
jLabel211.setText("b0");
jLabel211.setBounds(new Rectangle(616, 172, 12, 15));
jLabel212.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel212.setText("d0");
jLabel212.setBounds(new Rectangle(616, 216, 12, 15));
jLabel213.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel213.setText("c0");
jLabel213.setBounds(new Rectangle(616, 194, 12, 15));
jLabel214.setBounds(new Rectangle(619, 69, 28, 29));
jLabel214.setText("c1");
jLabel214.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField37.setBounds(new Rectangle(644, 95, 56, 19));
jTextField37.setText("0.00");
jTextField37.setBorder(BorderFactory.createLineBorder(Color.black));
jLabel215.setBounds(new Rectangle(619, 91, 30, 29));
jLabel215.setText("d1");
jLabel215.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel216.setBounds(new Rectangle(619, 46, 30, 29));
jLabel216.setText("b1");
jLabel216.setFont(new java.awt.Font("Dialog", 1, 10));
jTextField36.setBounds(new Rectangle(644, 72, 56, 19));
jTextField36.setText("0.50");
jTextField36.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField35.setBounds(new Rectangle(644, 49, 56, 19));
jTextField35.setText("0.40");
jTextField35.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField34.setBounds(new Rectangle(644, 26, 56, 19));
jTextField34.setText("0.10");
jTextField34.setBorder(BorderFactory.createLineBorder(Color.black));
jLabel217.setBounds(new Rectangle(619, 22, 30, 29));
jLabel217.setText("a1");
jLabel217.setFont(new java.awt.Font("Dialog", 1, 10));
jPanel13.setBorder(border20);
jPanel13.setBounds(new Rectangle(608, 2, 110, 124));
jTextField41.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField41.setText("0.00");
jTextField41.setBounds(new Rectangle(755, 99, 46, 19));
jLabel219.setFont(new java.awt.Font("Dialog", 1, 10));
jLabel219.setText("a2");
jLabel219.setBounds(new Rectangle(736, 28, 13, 15));
jTextField40.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField40.setText("0.20");
jTextField40.setBounds(new Rectangle(755, 73, 46, 19));
jTextField39.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField39.setText("0.83");
jTextField39.setBounds(new Rectangle(755, 49, 46, 19));
jTextField38.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField38.setText("-0.02");
jTextField38.setBounds(new Rectangle(755, 24, 46, 19));
jTextField29.setBorder(BorderFactory.createLineBorder(Color.black));
jTextField29.setText("0.00");
jTextField29.setBounds(new Rectangle(549, 364, 61, 19));
jLabel2112.setBounds(new Rectangle(396, 360, 143, 29));
jLabel2112.setText("Viscous Coupling Value");
jLabel2112.setFont(new java.awt.Font("Dialog", 1, 10));
saveInputData.setBounds(new Rectangle(662, 261, 124, 17));
saveInputData.setBorder(BorderFactory.createEtchedBorder());
saveInputData.setToolTipText("");
saveInputData.setActionCommand("Save Input Data");
saveInputData.setMnemonic('S');
saveInputData.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   saveInputData_actionPerformed(e);
  }
 });
 saveInputData.setText("Save Input Data");
 openInputData.setText("Open Input Data");
 openInputData.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   openInputData_actionPerformed(e);
  }
 });
 openInputData.setMnemonic('O');
 openInputData.setActionCommand("Save Input Data");
 openInputData.setToolTipText("");
 openInputData.setBorder(BorderFactory.createEtchedBorder());
 openInputData.setBounds(new Rectangle(662, 281, 124, 17));
 jProgressBar1.setBorder(border27);
 jProgressBar1.setToolTipText("Simulation run status");
 jProgressBar1.setMaximum(30);
 jProgressBar1.setBounds(new Rectangle(641, 385, 173, 10));
 jLabel220.setFont(new java.awt.Font("Dialog", 1, 10));
 jLabel220.setBorder(border28);
 jLabel220.setText("       Simulation Run Status");
 jLabel220.setBounds(new Rectangle(645, 375, 161, 13));
 jLabel2110.setBounds(new Rectangle(736, 50, 15, 15));
 jLabel2110.setText("b2");
 jLabel2110.setFont(new java.awt.Font("Dialog", 1, 10));
 jLabel2111.setBounds(new Rectangle(736, 74, 13, 15));
 jLabel2111.setText("c2");
 jLabel2111.setFont(new java.awt.Font("Dialog", 1, 10));
 jLabel2113.setBounds(new Rectangle(736, 100, 13, 15));
 jLabel2113.setText("d2");
 jLabel2113.setFont(new java.awt.Font("Dialog", 1, 10));
 jPanel14.setBounds(new Rectangle(720, 3, 97, 123));
 jPanel14.setBorder(border18);
 jPanel11.setBounds(new Rectangle(639, 240, 176, 131));
 jPanel11.setBorder(border11);
 implicitNewtonRaphson.setFont(new java.awt.Font("Dialog", 1, 10));
 implicitNewtonRaphson.setBorder(null);
 implicitNewtonRaphson.setSelected(false);
 implicitNewtonRaphson.setMnemonic('0');
 implicitNewtonRaphson.setText("N-Raphson ");
 implicitNewtonRaphson.setVisible(true);
 implicitNewtonRaphson.setBounds(new Rectangle(295, 255, 73, 13));
 implicitNewtonJacobi.setFont(new java.awt.Font("Dialog", 1, 10));
 implicitNewtonJacobi.setBorder(null);
 implicitNewtonJacobi.setSelected(true);
 implicitNewtonJacobi.setMnemonic('0');
 implicitNewtonJacobi.setText("N-Jacobi");
 implicitNewtonJacobi.setBounds(new Rectangle(295, 278, 73, 13));
 implicitNewtonJacobi.setVisible(true);
 jPanel15.setBorder(titledBorder29);
 jPanel15.setVisible(true);
 jPanel15.setBounds(new Rectangle(288, 234, 91, 68));
 jPanel15.setBorder(titledBorder28);
 jPanel16.setBorder(border37);
 jPanel16.setBounds(new Rectangle(383, 342, 247, 56));
 jif.getContentPane().add(jPanel1, null);
 jPanel1.add(jLabel1, null);
 jPanel1.add(jLabel2, null);
 jPanel1.add(jLabel3, null);
```

```
jPanel1.add(jLabel4, null);
jPanel1.add(jLabel5, null);
jPanel1.add(jLabel6, null);
jPanel1.add(jLabel7, null);
jPanel1.add(jTextField1, null);
jPanel1.add(jTextField3, null);
jPanel1.add(jTextField2, null);
jPanel1.add(jTextField4, null);
jPanel1.add(jTextField5, null);
jPanel1.add(jTextField6, null);
jPanel1.add(jTextField7, null);
jPanel1.add(jTextField13, null);
jPanel1.add(jTextField12, null);
jPanel1.add(jTextField11, null);
jPanel1.add(jTextField10, null);
jPanel1.add(jTextField9, null);
jPanel1.add(jTextField8, null);
jPanel1.add(jLabel9, null);
jPanel1.add(jLabel10, null);
jPanel1.add(jLabel11, null);
jPanel1.add(jLabel12, null);
jPanel1.add(jLabel13, null);
jPanel1.add(jLabel14, null);
jPanel1.add(jButton5, null);
jif.getContentPane().add(jPanel3, null);
jPanel6.setBounds(new Rectangle(4, 18, 184, 62));
jPanel6.setLayout(null);
jPanel6.add(jTextField22, null);
jPanel6.setBorder(titledBorder6);
jPanel6.add(jLabel22, null);
jPanel6.add(r12, null);
jPanel6.add(jRadioButton2, null);
jPanel3.add(jPanel7, null);
jPanel7.add(jSlider1, null);
jPanel7.add(jTextField23, null);
jPanel7.add(jLabel23, null);
jPanel3.add(jPanel6, null);
jif.getContentPane().add(jPanel2, null);
jPanel2.add(jLabel17, null);
jPanel2.add(jLabel18, null);
jPanel2.add(jLabel19, null);
jPanel2.add(jLabel20, null);
jPanel2.add(jLabel21, null);
jPanel2.add(jLabel16, null);
jPanel2.add(jLabel15, null);
jPanel2.add(jLabel8, null);
jPanel2.add(jTextField16, null);
jPanel2.add(jTextField14, null);
jPanel2.add(jTextField15, null);
jPanel2.add(jTextField17, null);
jPanel2.add(jTextField18, null);
jPanel2.add(jTextField19, null);
jPanel2.add(jTextField20, null);
jPanel2.add(jTextField21, null);
jPanel9.add(no, null);
jPanel9.add(yes, null);
jif.getContentPane().add(saveInputData, null);
jif.getContentPane().add(openInputData, null);
jif.getContentPane().add(clearAll, null);
jif.getContentPane().add(stopSimulation, null);
jif.getContentPane().add(runSimulation, null);
jif.getContentPane().add(jPanel11, null);
jif.getContentPane().add(jLabel220, null);
jif.getContentPane().add(jProgressBar1, null);
jif.getContentPane().add(viscousCoupling, null);
jif.getContentPane().add(capillaryCoupling, null);
jif.getContentPane().add(bothCouplings, null);
jif.getContentPane().add(noCoupling, null);
jif.getContentPane().add(jPanel5, null);
jif.getContentPane().add(jTextField39, null);
jif.getContentPane().add(jTextField40, null);
jif.getContentPane().add(jLabel219, null);
jif.getContentPane().add(jTextField41, null);
jif.getContentPane().add(jTextField38, null);
jif.getContentPane().add(jLabel2110, null);
jif.getContentPane().add(jLabel2111, null);
jif.getContentPane().add(jLabel2113, null);
jif.getContentPane().add(jPanel14, null);
jif.getContentPane().add(Fixed, null);
jif.getContentPane().add(Changeable, null);
InletSaturation.add(Fixed);
InletSaturation.add(Changeable);
jif.getContentPane().add(jLabel210, null);
jif.getContentPane().add(jLabel29, null);
jif.getContentPane().add(jTextField27, null);
jif.getContentPane().add(jTextField28, null);
jif.getContentPane().add(jPanel10, null);
jif.getContentPane().add(jTextField30, null);
jif.getContentPane().add(jLabel213, null);
jif.getContentPane().add(jTextField31, null);
jif.getContentPane().add(jTextField32, null);
jif.getContentPane().add(jTextField33, null);
jif.getContentPane().add(jLabel212, null);
jif.getContentPane().add(jLabel211, null);
jif.getContentPane().add(jLabel27, null);
jif.getContentPane().add(jPanel8, null);
jif.getContentPane().add(jTextField35, null);
jif.getContentPane().add(jLabel217, null);
jif.getContentPane().add(jTextField34, null);
jif.getContentPane().add(jLabel216, null);
jif.getContentPane().add(jLabel214, null);
jif.getContentPane().add(jTextField36, null);
jif.getContentPane().add(jTextField37, null);
jif.getContentPane().add(jLabel215, null);
jif.getContentPane().add(jPanel13, null);
jif.getContentPane().add(jPanel4, null);
jPanel4.add(jLabel24, null);
jPanel4.add(jLabel25, null);
jPanel4.add(jTextField24, null);
jPanel4.add(jTextField25, null);
jif.getContentPane().add(jPanel9, null);
R12.add(r12);
R12.add(jRadioButton2);
Flow.add(no);
Flow.add(yes);
Coupling.add(viscousCoupling);
Coupling.add(capillaryCoupling);
Coupling.add(bothCouplings);
Coupling.add(noCoupling);
jif.getContentPane().add(implicitNewtonJacobi, null);
jif.getContentPane().add(implicitNewtonRaphson, null);
jif.getContentPane().add(jPanel15, null);
```

```
  jif.getContentPane().add(jTextField29, null);
  jif.getContentPane().add(jLabel2112, null);
  jif.getContentPane().add(jPanel16, null);
  SolutionMethod.add(implicitNewtonRaphson);
  SolutionMethod.add(implicitNewtonJacobi);
  jif.getContentPane().add(jScrollPane1);
  jScrollPane1.setHorizontalScrollBarPolicy(JScrollPane. HORIZONTAL_SCROLLBAR_ALWAYS);
  jScrollPane1.setVerticalScrollBarPolicy(JScrollPane. VERTICAL_SCROLLBAR_ALWAYS);
  jif.setClosable(true);
  jif.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
  jif.setResizable(false);
  jif.setOpaque(true);
}

//Initiation of GUI ends here


// A method to set all textfields in the fluid properties box to null, i.e. " "
void jButton5_actionPerformed(ActionEvent e) {
  for (int i = 1; i < (13 + 1); i++) {
    jTextFieldArray[i].setText("");
  }
}

//A method to stop simulation run.
void stopSimulation_actionPerformed(ActionEvent e) {

  //Stop simulation run by setting isStop to true using a annonymous thread class
  if (isStop == false) {
    new Thread(new Runnable() {
      public void run() {
        isStop = true;
      }
    }).start();



    //Reset the progressbar to signify end of computation
    jProgressBar1.setIndeterminate(false);

    //Message box to signify end of simulation, when the simulation is interrupted
    JOptionPane.showMessageDialog(null,  "Simulation has been interrupted. Try again!");
  }
}

//A method to set all textfields to null, i.e. " "
void clearAll_actionPerformed(ActionEvent e) {
  for (int i = 1; i < (41 + 3); i++) {
    jTextFieldArray[i].setText("");
  }
}

//A method to obtain the current value of jSlider1 and put it in valueOfSlide
//and then display the current value of valueOfSlide in the jTextField23
void jSlider1_actionChange(ChangeEvent e) {
  valueOfSlide = jSlider1.getValue();
  jTextField23.setText(String.valueOf(valueOfSlide));
}

//A method to obtain the current value of jTextField23 and put it in valueOfSlide
//and then display the current value of valueOfSlide in the jSlider1.
```

```
//This is the reverse of the above method.
void jTextField23_actionPerformed(ActionEvent e) {
 valueOfSlide = Integer.parseInt(jTextField23.getText());
 jSlider1.setValue(valueOfSlide);
}

//Method to set value of a for R12 computation = 0.
//This option leads to R12 = 1, since R12 = (1 - a * (1-S))
//Also set the TextField holding a uneditable. In addition specify
//char = '0', to be saved as input data
void jRadioButton2_actionPerformed(ActionEvent e) {
 jTextField22.setText("0.00");
 jTextField22.setEnabled(false);
 r12Selection = '0';
}

// Method to set value of "a" for R12 computation != 0.
//That is R12 should be computed. This option also resets "a" = 0.05.
// Also set the TextField holding "a" editable. In addition specify
//char = '1', to be saved as input data
void r12_actionPerformed(ActionEvent e) {
 jTextField22.setText("0.05");
 jTextField22.setEnabled(true);
 r12Selection = '1';
}

// Method (Option) to enable real-time toggling of solution options (N-Jacobi or N-Raphson)
void no_actionPerformed(ActionEvent e) {
 implicitNewtonRaphson.setEnabled(true);
 implicitNewtonJacobi.setEnabled(true);
}

// Method (Option) to disenable real-time toggling of solution options (N-Jacobi or N-Raphson)
void yes_actionPerformed(ActionEvent e) {
 implicitNewtonRaphson.setEnabled(true);
 implicitNewtonJacobi.setEnabled(true);
}

//This method makes the TextFields holding the values of VisCou uneditable
// It also sent the content to 1
void noCoupling_actionPerformed(ActionEvent e) {
 jTextField29.setText("0.0");
 jTextField29.setEnabled(false);
 couplingSelection = 'n';
}

//The next 3 methods make the TextFields holding the values of VisCou editable
void bothCouplings_actionPerformed(ActionEvent e) {
 jTextField29.setEnabled(true);
 couplingSelection = 'b';
 jTextField29.setText("2.00");
}

void capillaryCoupling_actionPerformed(ActionEvent e) {
 jTextField29.setEnabled(false);
 couplingSelection = 'c';
 jTextField29.setText("0.0");
}

void viscousCoupling_actionPerformed(ActionEvent e) {
 jTextField29.setEnabled(true);
```

```
    couplingSelection = 'v';
    jTextField29.setText("2.00");
  }

  void relative_actionPerformed(ActionEvent e) {
    noCoupling.setSelected(true);
    jTextField29.setText("0.0");
    jTextField29.setEnabled(false);
    couplingSelection = 'n';
  }

  void Fixed_actionPerformed(ActionEvent e) {
    jTextField27.setText("0.0");
    jTextField27.setEnabled(false);
    insSelection = 'f';
  }

  void Changeable_actionPerformed(ActionEvent e) {
    jTextField27.setText("0.1");
    jTextField27.setEnabled(true);
    insSelection = 'c';
  }

  javax.swing.JLabel jLabel35 = new JLabel();
  javax.swing.JLabel jLabel36 = new JLabel();
  javax.swing.JLabel jLabel37 = new JLabel();
  javax.swing.JTextField jTextField44 = new JTextField();
  javax.swing.JTextField jTextField42 = new JTextField();
  javax.swing.JTextField jTextField43 = new JTextField();
  javax.swing.JPanel jPanel12 = new JPanel();
  javax.swing.border.Border border38 = javax.swing.BorderFactory.
      createEtchedBorder(Color.white, new java.awt.Color(134, 134, 134));
  javax.swing.border.Border border39 = new javax.swing.border.TitledBorder(border38, "Pc Fitting Type");
 javax.swing.border.Border border40 = javax.swing.BorderFactory.
      createEtchedBorder(Color.white, new java.awt.Color(134, 134, 134));
  javax.swing.border.Border border41 = new javax.swing.border.TitledBorder(border40, "Output Options");
 javax.swing.JPanel jPanel17 = new JPanel();
  javax.swing.border.Border border42 = javax.swing.BorderFactory.
      createEtchedBorder(Color.white, new java.awt.Color(134, 134, 134));
  javax.swing.border.Border border43 = new javax.swing.border.TitledBorder(border42, "Output Options");

  private void jbInit() throws Exception {

    jLabel35.setFont(new java.awt.Font("Dialog", 1, 10));
    jLabel35.setText("Pc Fit Eq. (1 or 2)");
    jLabel35.setBounds(new Rectangle(15, 24, 111, 22));
    jLabel36.setFont(new java.awt.Font("Dialog", 1, 10));
    jLabel36.setText("Print at Distance");
    jLabel36.setBounds(new Rectangle(15, 25, 111, 22));
    jLabel37.setFont(new java.awt.Font("Dialog", 1, 10));
    jLabel37.setText("Print at Timestep");
    jLabel37.setBounds(new Rectangle(15, 60, 111, 22));
    jTextField42.setBorder(BorderFactory.createLineBorder(Color.black));
    jTextField43.setBorder(BorderFactory.createLineBorder(Color.black));
    jTextField44.setBorder(BorderFactory.createLineBorder(Color.black));
    jTextField44.setText("2.0");
    jTextField44.setBounds(new Rectangle(146, 24, 46, 20));
    jTextField44.addActionListener(new IcsClass_jTextField44_actionAdapter(this));
    jTextField42.setText("1.000");
    jTextField42.setBounds(new Rectangle(146, 25, 46, 20));
    jTextField43.setText("1500");
    jTextField43.setBounds(new Rectangle(146, 60, 46, 20));
    jPanel12.setLayout(null);
    jPanel12.setFont(new java.awt.Font("Dialog", Font.BOLD, 13));
    jPanel12.setBorder(border39);
    jPanel12.setBounds(new Rectangle(6, 6, 216, 60));
    jPanel17.setFont(new java.awt.Font("Dialog", Font.BOLD, 13));
    jPanel17.setBorder(border43);
    jPanel17.setBounds(new Rectangle(6, 75, 216, 100));
    jPanel17.setLayout(null);
    ControlOption.getContentPane().setLayout(null);
    jPanel12.add(jLabel35);
    jPanel12.add(jTextField44);
    ControlOption.getContentPane().add(jPanel17, null);
    jPanel17.add(jLabel36);
    jPanel17.add(jLabel37);
    jPanel17.add(jTextField42);
    jPanel17.add(jTextField43);
    ControlOption.getContentPane().add(jPanel12, null);
  }

//CONSTRUCTION OF GUI ENDS HERE

//DECLARATION OF ALL PUBLIC VARIABLES STARTS HERE

 /** Variables (fields) to hold input from TextFields
  * nwDensity = non-wetting Density, wDensity = wetting Density, nwVis = non-wetting viscosity
  * wVis = wetting viscosity, injTime = injection time, wInjRate = wetting injection time
  * fluidIterationNumber =  number of time interval to compute fluid properties
  * effKnw = effective permeability to non-wetting at Swi (This is reference permeability for Kr)
  * effKw = effective permeability to wetting at Sor
  * Swi = initial or irreducible wetting saturation
  * Sor = residual or irreducible non-wetting saturation
  * wInjRate  = wetting injection Rate
  * flowTolerance = tolerance flow fractional flow solution convergence
  * flowCountTolerance = maximum number of iteration before exception is generated
  * w = wetting e.g. water, nw = non-wetting e.g. oil
  * l = Length, t = thickness, h = height, KroFitTypepv = fitting equation type, inc = inclination
  * KrwFitType = fitting equation type, Ac = Area under the capillary pressure curve
  * R12a = hydrodynamic effect factor, bNum = number of saturation grid/block
  * Swf = breakthrough saturation, Fwf = breakthrough fractional flow
  * VisCou = ViscousCoupling term
  * pca0, pca1, pca2, pca3 = Pc fitting coeeficients
  * krwa0, krwa1, n, krwa_R2, kroa0, kroa1, m, kroa_R2 = Kr fitting cofficients
  * Assumed_f = Initially assumed solution of Frac. Flow in each grid block at first time step
  * InitialF = Inlet Frac. Flow, inletS1 =Inlet saturation S* at time step n
  * inletS2 = Inlet saturation S* at time step n+1
  * deltaS1, deltaS2=inletS1/(bNum + 1), inletS2/(bNum + 1);
  * temp_inS1, temp_deltaS1=to save inletS1 or deltaS1 for the calculation of variable S*
  * epsV = MBE Check Ratio,
  * Increamental=value changed every time step for S*(inletS2-inletS1)
  * setUnit=1, used to determine Increamental value
  * counterG, counterL=used to determine Increamental value
  * switchCF=to check if the variable inlet stauration is switched to Fixed S*
  * Note: w = wetting (water) and o = non-wetting (oil)
  */

 double nwDensity, wDensity, nwVis, wVis, injTime, wInjRate, flowTolerance;
 int flowIterationNumber, bNum, flowCountTolerance;
 double effKnw, effKw, Swi, Sor;
 double l, t, h, porosity, inc, KrwFitType,KroFitType, Ac;
 double PcFitType,PrintDis,PrintTimeStep;
double R_12a, VisCou;
double pca0, pca1, pca2, pca3;
double krwa0, krwa1, m, krwa_R2, kroa0, kroa1, n, kroa_R2;
double InitialF, Assumed_f, InitialDis;
double epsV, inletS2, deltaS1, deltaS2;
static double inletS1, RunNum, setUnit;
boolean counterG, counterL, switchCF;
double Increamental;
double temp_inS1, temp_deltaS1;
double mr,nc,ng,velocity, deltaDensityRho, norTime, norFullTime, Time;

//Variable to save coupling selection option. The default data is viscous 'v'
char couplingSelection = 'v';

//Variable to select r12 option. The default data is do not select R12 '0'
char r12Selection = '0';
char insSelection = 'c';

// A public variable to be used by jSlider1_actionChange and jTextField23_actionPerformed
int valueOfSlide;

//A static variable to count the number of new "input data"
static int inputNumber = 0;

//A static variable to count the number of Flow Solution Iteration"
static int flowCount = 0;

//Array to hold computed distance travelled
double[] normalizedDis = new double[bNum + 2];

//Array to hold computed the derivative of distance with respect to Saturation
double[] dd_ds = new double[bNum + 2];

//Array to hold the saturation value for every grid points
double[] sat1 = new double[bNum + 2];
double[] temp_sat1 = new double[bNum + 2];
double[] sat2 = new double[bNum + 2];

//Array to hold the soluation of the matrix
double[] MatrixSolution = new double[bNum];

//Array to hold the soluation of fractional flow
double[] flowSolution = new double[bNum];

//Array to hold the soluation of fractional flow redistributed on the S* domain
double[] interplationF = new double[bNum];

//Array to hold the soluation of Normalized Distance redistributed on the S* domain
double[] interplationX = new double[bNum + 2];

//Arrays to hold computed pressure gradient
double[] norPressPotenOne = new double[bNum + 2];
double[] norPressPotenTwo = new double[bNum + 2];

//Variable for residual value computation
double MaxAbsMatrixSolution;

//Variable for monitoring simulation time interval
double initialTime, finalTime;

//Variables to hold MBE values
```

```
double epsResult, mbeResult;

//Variable for time interval division
static int timeDivision;

//Variable to count the number of new "output results"
static int ouputNumber = 0;

//Copies of the solution Matrix for iterative purpose or calculation of variable S*
double[] copyE = new double[bNum];
double[] copyETime = new double[bNum];
double[] tempETime = new double[bNum];
double[] copyETimeTime = new double[bNum];
double[] xettaOfSS = new double[bNum];
double[] copyDisTime = new double[bNum + 2];
double[] tempDis = new double[bNum + 2];
double[] fOfSS = new double[bNum + 2];
double[] gOfSS = new double[bNum + 2];
double[] cOfSS = new double[bNum + 2];

double[] krwValue = new double[bNum + 2];
double[] kroValue = new double[bNum + 2];
double[] lamdaOne = new double[bNum + 2];
double[] lamdaTwo = new double[bNum + 2];
double[] rOneTwo = new double[bNum + 2];
double[] lamdaM = new double[bNum + 2];

double[] alphaOne = new double[bNum + 2];
double[] alphaTwo = new double[bNum + 2];

double[] deltaPlc = new double[bNum + 2];
double[] couplingTerm = new double[bNum + 2];

//Create JTextField Array for opening save input data
JTextField[] jTextFieldArray = {
  jTextField1, jTextField1, jTextField2, jTextField3, jTextField4,
  jTextField5, jTextField6, jTextField7, jTextField8, jTextField9,
  jTextField10, jTextField11, jTextField12, jTextField13, jTextField14,
  jTextField15, jTextField16, jTextField17, jTextField18, jTextField19,
  jTextField20, jTextField21, jTextField22, jTextField23, jTextField24,
  jTextField25, jTextField27, jTextField28, jTextField29, jTextField30,
  jTextField31, jTextField32, jTextField33, jTextField34, jTextField35,
  jTextField36, jTextField37, jTextField38, jTextField39, jTextField40,
  jTextField41,jTextField42, jTextField43,jTextField44};

//DECLARATION OF ALL PUBLIC VARIABLES STOPS HERE

//IMPLEMENTATION OF "Normalized and other variables" STARTS HERE
public double mr() {
  double mr = ( (effKw * nwVis) / (effKnw * wVis));
  return mr;
}

public double nc() {
  double nc = (Ac * (effKw / wVis)) / (velocity * l);
  return nc;
}

public double ng() {
  double ng = ( ( (effKw / wVis) * deltaDensityRho * 9.81) * Math.sin( (Math.PI * inc / 180))) / velocity;
  return ng;
```

```
 }

 public double deltaDensityRho() {
  double deltaDensityRho = (wDensity - nwDensity);
  return deltaDensityRho;
 }

 public double[] rOneTwo() {
  double[] A = new double[bNum + 2];
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = (1 - (R_12a * (1 - sat1[i])));
   }
   return A;
 }
 public double[] lamdaOne() {

  double[] A = new double[bNum + 2];
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = (krwValue[i] * effKw / wVis);
   }
   return A;
 }

 public double[] lamdaTwo() {

  double[] A = new double[bNum + 2];
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = (kroValue[i]* effKnw / nwVis);
   }
   return A;
 }


 public double[] lamdaM() {
   double[] A = new double[bNum + 2];
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = (sat1[i]* (effKw / wVis)) + ( (1 - sat1[i]) * (effKnw / nwVis));
   }
   return A;
 }

 public double[] alphaOne() {
  double[] A = new double[bNum + 2];

  for (int i = 0; i < bNum + 2; i++) {
   //No coupling
   if (noCoupling.isSelected()) {
    A[i] = 1;
   }
   //Only capillary coupling
   else if (capillaryCoupling.isSelected()) {
    A[i] = (1 - porosity);
   }
   //Only Viscous coupling
   else if (viscousCoupling.isSelected()) {
    A[i] = ( (1 - ( (VisCou / rOneTwo[i]) *(lamdaTwo[i] / lamdaM[i]))));
   }
   //Both viscous and capillary coupling
   else if (bothCouplings.isSelected()) {
    A[i] = ( ( (1 - porosity) * ( (1 - (VisCou / rOneTwo[i]) * (lamdaTwo[i] /lamdaM[i])))));
   }
```

```
    }
    return A;
  }

  public double[] alphaTwo() {
    double[] A = new double[bNum + 2];

    for (int i = 0;  i < bNum + 2; i++) {
      //No coupling
      if (noCoupling.isSelected()) {
        A[i] = 1 ;
      }
      //Only capillary coupling
      else if (capillaryCoupling.isSelected()) {
        A[i] = (1  - porosity);
      }
      //Only Viscous coupling
      else if (viscousCoupling.isSelected()) {
        A[i] = ( (1  - (VisCou * rOneTwo[i]) *(lamdaOne[i] / lamdaM[i])));
      }
      //Both viscous and capillary coupling
      else if (bothCouplings.isSelected()) {
        A[i] = ( (1  - porosity) * ( (1 - (VisCou * rOneTwo[i]) * (lamdaOne[i] /lamdaM[i]))));
        }
    }
    return A;
  }

  public double[] fOfS() {
    double[] A = new double[bNum + 2];

    for (int i = 0; i < bNum + 2; i++) {
      if (noCoupling.isSelected() && R_12a == 0) {
        A[i] = (mr * krwValue[i]) /(mr * krwValue[i] + kroValue[i]);
      }
      else {
        A[i] = (rOneTwo[i] * mr * krwValue[i]) / (rOneTwo[i] * mr * krwValue[i] + kroValue[i]);
      }
    }
    return A;
  }

  public double[] gOfS() {
    double[] A = new double[bNum + 2];

    for (int i = 0; i < bNum + 2; i++) {
      if (noCoupling.isSelected() && R_12a == 0) {
        A[i] = (1 - (ng * kroValue[i] / mr)) * fOfSS[i];
      }
      else {
        A[i] = (1 - (couplingTerm[i] * ( (1 + ( (nwDensity / deltaDensityRho) *
        (1 - rOneTwo[i]))) *ng * kroValue[i]) /(rOneTwo[i] * mr))) * fOfSS [i];
      }
    }
    return A;
  }

  public double[] cOfS() {
    double[] A = new double[bNum + 2];

    for (int i = 0; i < bNum + 2; i++) {
```

```
    if (noCoupling.isSelected() && R_12a == 0) {
     A[i] = - ( (1 / mr) * fOfSS [i] * kroValue[i]*deltaPIc[i]);
    }
    else {
     A[i] = ( -1 / (mr * rOneTwo[i])) * (couplingTerm[i] * fOfSS [i] * kroValue[i] * deltaPIc[i]);
    }
   }
   return A;
  }

  public double norTime() {
   double norTime = (velocity * injTime) / (porosity * l * (1 - Sor - Swi));
   return norTime;
  }

  public double norFullTime() {
   double norFullTime = (norTime / flowIterationNumber);
   return norFullTime;
  }

  public double Time() {
   double Time = timeDivision * norFullTime * (porosity * l * (1 - Sor - Swi)) / velocity;
   return Time;
  }

  public double[] xettaOfS() {
   double[] A = new double[bNum];
   for (int i = 0; i < bNum; i++) {
    A[i] = norFullTime / (nc * Math.pow(deltaS1, 2) * cOfSS[i + 1]);
   }
   return A;
  }

  public double velocity() {
   double velocity = (wInjRate) / (t * h);
   return velocity;
  }

  public double deltaS(double inS) {
   return (0.99999999999 / (bNum + 1)) * inS;
  }

//IMPLEMENTATION OF "Normalized and other Variables" ENDS HERE


//IMPLEMENTATION OF "Kr", "Coupling Term" and "deltaPIc" STARTS HERE

  public double[] deltaPIc() {

   double[] A = new double[bNum + 2];

    if(PcFitType==2.0){
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = (1 / Ac) * (3 * pca0 * sat1[i] * sat1[i] + 2 * pca1 * sat1[i]+ pca2);
   }
  }
  else if(PcFitType==1.0){
   for (int i = 0; i < bNum + 2; i++) {
    A[i]  = (1 / Ac) *( ( ( ( ((1 + pca2 * sat1[i] + pca3 * Math.pow(sat1[i], 2))
          * ( -pca0 - 2 * pca1 * sat1[i]))) -( ( (pca0 * (1 - sat1[i]) +
         pca1 * (1 - Math.pow(sat1[i], 2)))* (pca2 + 2 * pca3 * sat1[i]))))/
         ( (1 + pca2 * sat1[i] + pca3 * Math.pow(sat1[i], 2)) *
         (1 + pca2 * sat1[i] + pca3 *Math.pow(sat1[i], 2))));
  }
  }

  return A;
 }


 public double[] couplingTerm() {

  double[] A = new double[bNum + 2];
  for (int i = 0; i < bNum + 2; i++) {
   A[i] = ( (alphaOne[i] + alphaTwo[i]) / 2);
  }
  return A;
 }

 public double[] krwValue() {
  double[] A = new double[bNum + 2];
  if(KrwFitType==1.0){
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = ( (krwa0 + krwa1 * (1 - sat1[i])) / (krwa0 + (1 - sat1[i])))*Math.pow(sat1[i], m);
   }
  }
  else{
   for (int i = 0; i < bNum + 2; i++) {
    A[i] = krwa0 * sat1[i]+ krwa1 * Math.pow(sat1[i], 2) + m * Math.pow(sat1[i], 3)+krwa_R2*Math.pow(sat1[i],
4);
   }
  }
  return A;
 }

 public double[] kroValue() {
  double[] A = new double[bNum + 2];
  if(KrwFitType==1.0){
  for (int i = 0; i < bNum + 2; i++) {
   A[i] = ( (kroa0 + kroa1 * sat1[i]) / (kroa0 + sat1[i])) *Math.pow( (1 - sat1[i]), n);
   }
  }
  else{
  for (int i = 0; i < bNum + 2; i++) {
   A[i] =kroa0 * (1 -sat1[i]) + kroa1 * Math.pow( (1 - sat1[i]), 2) +
       n * Math.pow( (1 - sat1[i]), 3) + kroa_R2 * Math.pow( (1 - sat1[i]), 4);
   }
  }
  return A;
 }



 //IMPLEMENTATION OF "Kr", "Coupling Term" and "deltaPIc" ENDS HERE

 //IMPLEMENTATION OF "Flow Coefficients, distance and pressure computations" STARTS HERE
 public double[] Di(double[] gOfS, double[] xettaOfS) {
  int j = bNum;
  double[] A = new double[j];

  //Populate A
  for (int i = 0; i < j; i++) {
    if (timeDivision == 1) {
      if (flowCount == 1) {
        if (i == 0) {
          A[i] = (Math.pow( (Assumed_f - gOfS[i + 1]), 2) *
                (Assumed_f - 2 * Assumed_f + 0) * xettaOfS[i] - Assumed_f +InitialF);
        }
        else if (i > 0 && i < (j - 1)) {
          A[i] = (Math.pow( (Assumed_f - gOfS[i + 1]), 2) *
                  (Assumed_f - 2 * Assumed_f + Assumed_f) * xettaOfS[i] - Assumed_f + InitialF);
          }
        else if (i == (j - 1)) {
          A[i] = (Math.pow( (Assumed_f - gOfS[i + 1]), 2) *
                  (1 - 2 * Assumed_f + Assumed_f) * xettaOfS[i] - Assumed_f + InitialF);
          }
       }
      else if (flowCount > 1) {
        if (i == 0) {
          A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                  (copyE[i + 1] - 2 * copyE[i] + 0) * xettaOfS[i] - copyE[i] + InitialF);
          }
        else if (i > 0 && i < (j - 1)) {
          A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                  (copyE[i + 1] - 2 * copyE[i] + copyE[i - 1]) * xettaOfS[i] - copyE[i] + InitialF);
          }
        else if (i == (j - 1)) {
          A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                  (1 - 2 * copyE[i] + copyE[i - 1]) * xettaOfS[i] - copyE[i] +  InitialF);
          }
       }
     }
    else if (timeDivision > 1) {
      if (i == 0) {
        A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                (copyE[i + 1] - 2 * copyE[i] + 0) * xettaOfS[i] - copyE[i] + copyETime[i]);
        }
      else if (i > 0 && i < (j - 1)) {
        A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                (copyE[i + 1] - 2 * copyE[i] + copyE[i - 1]) * xettaOfS[i] -  copyE[i] + copyETime[i]);
       }
      else if (i == (j - 1)) {
        A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) *
                (1 - 2 * copyE[i] + copyE[i - 1]) * xettaOfS[i] - copyE[i] + copyETime[i]);
       }
     }
   }
  return A;
}

public double[] dFplusOne(double[] gOfS, double[] xettaOfS) {
  int j = bNum - 1;
  double[] A = new double[j];

  //Populate A
  for (int i = 0; i < j; i++) {
    if (timeDivision == 1 && flowCount == 1) {
      A[i] = (Math.pow( (Assumed_f - gOfS[i + 1]), 2) * xettaOfS[i]);
     }
    else {
      A[i] = (Math.pow( (copyE[i] - gOfS[i + 1]), 2) * xettaOfS[i]);
     }
   }
  return A;
 }

 public double[] dFminusOne(double[] gOfS, double[] xettaOfS) {
  int j = bNum - 1;
  double[] A = new double[j];

  //Populate A
  for (int i = 0; i < j; i++) {
   if (timeDivision == 1 && flowCount == 1) {
    A[i] = (Math.pow( (Assumed_f - gOfS[i + 2]), 2) * xettaOfS[i + 1]);
   }
   else {
    A[i] = (Math.pow( (copyE[i + 1] - gOfS[i + 2]), 2) * xettaOfS[i + 1]);
   }
  }
  return A;
 }

 public double[] dFnone(double[] gOfS, double[] xettaOfS) {
  int j = bNum;
  double[] A = new double[j];

  //Populate A
  for (int i = 0; i < j; i++) {
   if (timeDivision == 1 && flowCount == 1) {

    if (i == 0) {
     A[i] = (2 * xettaOfS[i] * (Assumed_f - gOfS[i + 1]) * (Assumed_f - 3 * Assumed_f + 0 + gOfS[i + 1])) - 1;
    }
    else if (i > 0 && i < j - 1) {
     A[i] = (2 * xettaOfS[i] * (Assumed_f - gOfS[i + 1]) *
          (Assumed_f - 3 * Assumed_f + Assumed_f + gOfS[i + 1])) - 1;
    }
    else if (i == j - 1) {
     A[i] = (2 * xettaOfS[i] * (Assumed_f - gOfS[i + 1]) * (1 - 3 * Assumed_f + Assumed_f + gOfS[i + 1])) - 1;
    }
   }
   else {
    if (i == 0) {
     A[i] = (2 * xettaOfS[i] * (copyE[i] - gOfS[i + 1]) *(copyE[i + 1] - 3 * copyE[i] + 0 + gOfS[i + 1])) - 1;
    }
    else if (i > 0 && i < j - 1) {
     A[i] = (2 * xettaOfS[i] * (copyE[i] - gOfS[i + 1]) *
          (copyE[i + 1] - 3 * copyE[i] + copyE[i - 1] + gOfS[i + 1])) -1;
    }
    else if (i == j - 1) {
     A[i] = (2 * xettaOfS[i] * (copyE[i] - gOfS[i + 1]) * (1 - 3 * copyE[i] + copyE[i - 1] + gOfS[i + 1])) - 1;
    }
   }
  }
  return A;
 }

//Normalized distance travelled from frontal advance equation for fixed S*
public double[] zetta(double f[], double ff[], double PrevDis[], double s, int timeDivisionn) {

  int j = bNum + 2;
  double[] A = new double[j];
  double c = norFullTime / s / 2; //ratio of deltaTime (norFullTime) to deltaS
```

```
   //Compute individual grid distance based on flow

   if (timeDivisionn == 1) {
    A[j - 1] = ( (1 - f[j - 3]) + (1 - ff[j - 3])) * c + InitialDis;
    for (int i = j - 2; i > 1; i--) {
     A[i] = ( (f[i - 1] - f[i - 2]) + (ff[i - 1] - ff[i - 2])) * c +InitialDis;
    }
    A[1] = ( (f[0] - 0) + (ff[0] - 0)) * c + InitialDis;
    A[0] = 2 * A[1] - A[2];
   }

   else if (timeDivisionn > 1) {
    A[j - 1] = ( (1 - f[j - 3]) + (1 - ff[j - 3])) * c + PrevDis[j - 1];
    for (int i = j - 2; i > 1; i--) {
     A[i] = ( (f[i - 1] - f[i - 2]) + (ff[i - 1] - ff[i - 2])) * c +PrevDis[i];
    }
    A[1] = ( (f[0] - 0) + (ff[0] - 0)) * c + PrevDis[1];
    A[0] = 2 * A[1] - A[2];
   }
   return A;
 }

 //the derivative of distance with respect to Saturation for Fixed S* solution
 public double[] differ_dsF(double[] distance, double deltaS) {

   double[] A = new double[bNum + 2];

   for (int i = 0; i < bNum + 1; i++) {
    A[i] = - (distance[i] - distance[i + 1]) / deltaS;
   }
   A[bNum + 1] = (2 * A[bNum] - A[bNum - 1]);

   return A;
 }

 //the derivative of distance with respect to Saturation for Varialbe S* solution
 public double[] differ_dsC(double[] f, double[] cs, double[] gs, double nc, double inlet_s) {

   double[] A = new double[bNum + 2];

   for (int i = 1; i < bNum + 1; i++) {
    A[i] = -nc * cs[i] / (f[i - 1] - gs[i]);
   }
   if (inlet_s < 1.0) {
    //A[bNum + 1] = 2*A[bNum]-A[bNum-1];
    A[bNum + 1] = -nc * cs[bNum + 1] / (1.0 - gs[bNum + 1]);
   }
   else if (inlet_s == 1.0) {
    A[bNum + 1] = 2 * A[bNum] - A[bNum - 1];
   }
   A[0] = 2 * A[1] - A[2];

   return A;
 }

 //Normalized distance travelled from fractional flow equation for Variable S*
 public double[] dis(double[] differ_s, double delts) {

   double[] A = new double[bNum + 2];
   //Compute individual grid distance based on differ_s
   for (int i = 0; i < bNum + 1; i++) {
    for (int j = i; j < bNum + 1; j++) {
      A[i] = A[i] - delts * (differ_s[j] + differ_s[j + 1]) / 2;
    }
  }
  if (delts * (bNum + 1) / 0.99999999999 < 1.0) {
    A[bNum + 1] = 0;
  }
  else {
    A[bNum + 1] = A[bNum] + differ_s[bNum + 1] * delts;
  }
  return A;
}

//potential gradient in the wetting phase
public double[] potentialGradientOne(double[] fSol, double s, double[] ddds) {
  // fSol = fractional flow
  // ddds = partial differential Normalized distance with respect to Normalized Saturation
  // s = deltaS
  double[] E = new double[bNum + 2]; // Potential gradient in the wetting phase
  double A3 = 0;
  double A4 = 0;

  if (noCoupling.isSelected()) {
    for (int i = 1; i < bNum + 1; i++) {
      E[i] = -1 * (velocity * fSol[i - 1]) / lamdaOne[i];
    }
    E[bNum + 1] = -1 * (velocity * 1) / lamdaOne[(bNum + 1)];
  }
  else {
    for (int i = 1; i < bNum + 1; i++) {
      A3 = - (velocity * fSol[i - 1]) / lamdaOne[i];
      A4 = ( (1 - alphaOne[i]) / 2) * ( (deltaPIc[i] * Ac / (1 * ddds[i])) -
          ( (wDensity -rOneTwo[i] * nwDensity) *9.81 *Math.sin( (Math.PI * inc / 180))));
      E[i] = 1 * (A3 - A4);
    }
    A3 = - (velocity * 1) / lamdaOne[(bNum + 1)];
    A4 = ( (1 - alphaOne[bNum + 1]) / 2) * ( (deltaPIc[(bNum + 1)]* Ac / (1 * ddds[bNum + 1])) -
          ( (wDensity - rOneTwo[(bNum + 1)]* nwDensity) * 9.81 * Math.sin( (Math.PI * inc / 180))));
    E[bNum + 1] = 1 * (A3 - A4);
  }
  E[0] = 2 * E[1] - E[2];
  return E;
}

//potential gradient in the non-wetting phase
public double[] potentialGradientTwo(double[] fSol, double s, double[] ddds) {
  // fSol = fractional flow
  // ddds = partial differential Normalized distance with respect to Normalized Saturation
  // s = deltaS

  double[] E = new double[bNum + 2]; // Potential gradient in the wetting phase
  double A3 = 0;
  double A4 = 0;

  if (noCoupling.isSelected()) {
    E[0] = -1 * (velocity * (1 - 0)) / lamdaTwo[0];
    for (int i = 1; i < bNum + 1; i++) {
      E[i] = -1 * (velocity * (1 - fSol[i - 1])) / lamdaTwo[i];
    }
  }
  else {
      E[0] = 1 *( - (velocity * (1 - 0)) / lamdaTwo[0] + ( (1 - alphaTwo[0]) / (2 * rOneTwo[0])) *
          ( (deltaPIc[0] * Ac / (1 * ddds[0])) - ( (wDensity - rOneTwo[0] * nwDensity) * 9.81 * Math.sin( (Math.PI *
  inc / 180)))));
      for (int i = 1; i < bNum + 1; i++) {
        A3 = - (velocity * (1 - fSol[i - 1])) / lamdaTwo[i];
        A4 = ( (1 - alphaTwo[i]) / (2 * rOneTwo[i])) * ( (deltaPIc[i] * Ac / (1 * ddds[i])) -
            ( (wDensity - rOneTwo[i] * nwDensity) * 9.81 * Math.sin( (Math.PI * inc / 180))));
       E[i] = 1 * (A3 + A4);
      }
    }
    E[bNum + 1] = 2 * E[bNum] - E[bNum - 1];
    return E;
  }

  //Discretized saturation for dynamic viewing
  public double[] sat(double delta_s) {
    double[] sat = new double[bNum + 2];
    for (int i = 0; i < bNum + 2; i++) {
      sat[i] = i * delta_s;
    }
    return sat;
  }

//IMPLEMENTATION OF "Flow Coefficients, distance and pressure computations" ENDS HERE


//A METHOD FOR COPYING ARRAY STARTS HERE
  public double[] copy(double[] A) {
    int j = A.length;
    double[] E = new double[j];
    for (int i = 0; i < j; i++) {
      E[i] = A[i];
    }
    return E;
  }

//A METHOD FOR COPYING ARRAY STOPS HERE

  //Thomas algorithm method
  double[] thomasAlgorithmSolution(double upperElements[],double diagonalElements[],
                          double lowerElements[], double rhsElements[]) {

    //For an m x n tri-diagonal matrix, number of rows (m) = number of cols n = (matrix size)
    int matrixSize = diagonalElements.length;

    //Arrays to hold (primary) main computed coefficients,
    //secondary computed coefficients and solutions, x
    double A[] = new double[matrixSize]; //Primary
    double B[] = new double[matrixSize]; //Primary
    double C[] = new double[matrixSize]; //Primary
    double D[] = new double[matrixSize]; //Primary
    double W[] = new double[matrixSize]; //Secondary
    double G[] = new double[matrixSize]; //Secondary
    double E[] = new double[matrixSize]; //Solution
    double pivot = 0; //pivoting value

    //Check for zero pivot element

    if (diagonalElements[0] == 0) {
      //Set isStop to true to stop the loop
      isStop = true;
    //Reset the progressbar to signify end of computation
    jProgressBar1.setIndeterminate(false);
    //Give a message box to signify exception
    JOptionPane.showMessageDialog(null,
     "Zero Pivot Error 1 ! Change Time Step or Grid Number to Rectify.");
  }

  for (int i = 0; i < matrixSize; i++) {
    //Primary
    if (i == 0) {
      A[i] = 0;
    }
    else {
      A[i] = lowerElements[i - 1];
    }
    B[i] = diagonalElements[i];
    if (i == (matrixSize - 1)) {
      C[i] = 0;
    }
    else {
      C[i] = upperElements[i];
    }
    D[i] = rhsElements[i];

    //Secondary
    if (i == 0) {
      W[i] = (C[i] / B[i]);
      G[i] = (D[i] / B[i]);
    }
    else if (i > 0) {

      //Pivot condition testing
      pivot = (B[i] - (A[i] * W[i - 1]));
      if (pivot == 0) {
      //Reset the progressbar to signify end of computation
       jProgressBar1.setIndeterminate(false);
        //Give a message box to signify exception
        JOptionPane.showMessageDialog(null,
        "Zero Pivot Error 2 ! Change Time Step or Grid Number to Rectify.");
      }
      W[i] = (C[i] / (B[i] - (A[i] * W[i - 1])));
      G[i] = ( (D[i] - (A[i] * G[i - 1])) / (B[i] - (A[i] * W[i - 1])));
    }
  }

  //Solution
  for (int i = 0; i < matrixSize; i++) {
    int j = (matrixSize - 1) - i;

    if (i == 0) {
      E[j] = G[j];
    }
    else if (i > 0) {
      E[j] = (G[j] - (W[j] * E[j + 1]));
    }
  }
  return E;
 }

//A METHOD USED IN COMPUTING NAIVE-JACOBI-NEWTON SOLUTION STARTS HERE
 public double[] newtonJacobi(double[] A, double[] B) {
    int j = A.length;
    double[] E = new double[j];
    for (int i = 0; i < j; i++) {
      E[i] = A[i] / B[i];
    }
    return E;
  }

 //A METHOD USED IN COMPUTING NAIVE-JACOBI-NEWTON SOLUTION STOPS HERE

 //A METHOD FOR FLOW ACTUAL SOLUTION STARTS HERE

  public double[] flowActualSolution(double[] A, int timeDivision) {
    // A = DELTA f, which is the solution array of matrix
    int j = A.length;
    double[] E = new double[j];
    double initialAssumption = Assumed_f;
    if (timeDivision == 1 && flowCount == 1) {
      for (int i = 0; i < j; i++) {
        E[i] = initialAssumption - A[i];
      }
    }
    else {
      for (int i = 0; i < j; i++) {
        E[i] = copyE[i] - A[i];
      }
    }
    return E;
  }

 //A METHOD FOR FLOW ACTUAL SOLUTION ENDS HERE

 //A METHOD FOR absolute INDIVIDUAL Delta f(matrix solution) COMPUTATION STARTS HERE

  public double MaxAbsValue(double[] A) {

    int j = A.length;
    double[] R = new double[j];

    for (int i = 0; i < j; i++) {
      R[i] = Math.abs(A[i]);
    }
    return maximumOfArrayValues(R);
  }

 //A METHOD FOR absolute INDIVIDUAL Delta f(matrix solution) COMPUTATION STARTS HERE

 //A METHOD FOR SUM DISTANCE COMPUTATION STARTS HERE
  public double sumDis(double[] distance) {
    int j = distance.length;
    double sumDis = 0;

    for (int i = 0; i < j - 1; i++) {
      sumDis = sumDis + (distance[i] + distance[i + 1]) / 2;
    }
    return sumDis;
  }

 //A METHOD FOR SUM DISTANCE COMPUTATION ENDS HERE
```

```
//A METHOD FOR EPS(mateial balance check ratio) COMPUTATION STARTS HERE

  public double calculateEps(double sumD) {

    double calculateEps = (norFullTime * timeDivision - sumD * deltaS1) / norFullTime;
    return calculateEps;
  }

//A METHOD FOR EPS (mateial balance check ratio) COMPUTATION ENDS HERE

//A METHOD FOR MBE COMPUTATION STARTS HERE

  public double calculateMBE(double sumD) {

    double calculateMBE = Math.abs(norFullTime * timeDivision - sumD * deltaS1) /(norFullTime *
timeDivision);

    return calculateMBE;
  }

//A METHOD FOR MBE COMPUTATION ENDS HERE


//A METHOD FOR INTERPOLATION OF FRACTIONAL FLOW SOLUTION STARTS HERE.

  public double[] interpolateF(double in_S, double[] sat2, double[] fflow1) {
    // in_S = Current inlet saturation
    // fflow1 = Fractional flow solution from in_S
    // sat2 =[0, bnum+1] array of Discretized saturation based on Next assumed inlet saturation
    // fflow2 = Redistribution of fflow1 on the new domain sat2
    double[] fflow2 = new double[bNum];
    double delta_S = deltaS(in_S);
    double[] sat_S = sat(delta_S);

    for (int i = 0; i < bNum; i++) {
      if (sat2[i + 1] < in_S) {
        double y = sat2[i + 1] / delta_S;
        int N = (int) y;
        if (N == 0) {
          fflow2[i] = 0 +(fflow1[N] - 0) / (sat_S[N + 1] - sat_S[N]) * (sat2[i + 1] - sat_S[N]); (sat2[i + 1] - sat_S[N]);
        }
        else if (N > 0 && N < bNum) {
          fflow2[i] = fflow1[N - 1] + (fflow1[N] - fflow1[N - 1]) / (sat_S[N + 1] - sat_S[N]) * (sat2[i + 1] - sat_S[N]);
         }
        else if (N == bNum) {
          fflow2[i] = fflow1[N - 1] +(1.0 - fflow1[N - 1]) / (sat_S[N + 1] - sat_S[N]) * (sat2[i + 1] - sat_S[N]);
        }
      }
      else if (sat2[i + 1] >= in_S) {
        fflow2[i] = 1.0;
      }
    }
    return fflow2;
  }

//A METHOD FOR INTERPOLATION OF FRACTIONAL FLOW SOLUTION STARTS HERE.

//A METHOD FOR INTERPOLATION OF FRACTIONAL FLOW SOLUTION STARTS HERE.

  public double[] interpolateX(double in_S, double[] sat2, double[] dis1) {
    // in_S = Current inlet saturation
```

```
   // dis1 = Normalized distance from in_S
   // sat2 =[0, bnum+1] array of Discretized saturation based on Next assumed inlet saturation
   // dis2 = Redistribution of dis1 on the new domain sat2
   double[] dis2 = new double[bNum + 2];
   double delta_S = deltaS(in_S);
   double[] sat_S = sat(delta_S);

   dis2[0] = dis1[0];
   for (int i = 1; i < bNum + 2; i++) {

    double y = sat2[i] / delta_S;
    int N = (int) y;
    if (N < bNum + 1) {
     dis2[i] = dis1[N] +(dis1[N + 1] - dis1[N]) / (sat_S[N + 1] - sat_S[N]) * (sat2[i] - sat_S[N]);
    }
    else {
     dis2[i] = 0.0;
    }
   }
   return dis2;
  }

//A METHOD FOR INTERPOLATION OF FRACTIONAL FLOW SOLUTION STARTS HERE.


//A METHOD FOR COEFFICIENTS COMPUTATION STARTS HERE.

  public void calculateCoef(double inletS1) {

   deltaS1 = deltaS(inletS1);
   sat1 = sat(deltaS1);

   krwValue=krwValue();
   kroValue=kroValue();
   lamdaOne=lamdaOne();
   lamdaTwo=lamdaTwo();
   rOneTwo=rOneTwo();
   lamdaM=lamdaM();
   alphaOne=alphaOne();
   alphaTwo=alphaTwo();
   deltaPlc=deltaPlc();
   couplingTerm=couplingTerm();

   fOfSS = fOfS();
   gOfSS = gOfS();
   cOfSS = cOfS();
   xettaOfSS = xettaOfS();

   return;
  }

  public void calBaseCoef() {
    velocity=velocity();
    deltaDensityRho=deltaDensityRho();
    norTime=norTime();
    norFullTime=norFullTime();
    mr = mr();
    nc = nc();
    ng = ng();

    return;
```

```
 }

//A METHOD FOR COEFFICIENTS COMPUTATION STARTS HERE.

//A METHOD FOR RETURNING MAXIMUM VALUE FROM A 1-D ARRAY OF DOUBLE VALUE STARTS
HERE

 static double maximumOfArrayValues(double[] A) {
  double staticMaximumValue = 0;
  double dynamicMaximumValue = 0;
  for (int i = 0; i < (A.length - 1); i++) {
   if (i == 0) {
    if (A[i] >= A[i + 1]) {
     dynamicMaximumValue = A[i];
     staticMaximumValue = dynamicMaximumValue;
    }
    else {
     dynamicMaximumValue = A[i + 1];
     staticMaximumValue = dynamicMaximumValue;
    }
   }
   else if (i > 0) {
    if (A[i] >= A[i + 1]) {
     dynamicMaximumValue = A[i];
     staticMaximumValue = Math.max(staticMaximumValue, dynamicMaximumValue);
    }
    else {
     dynamicMaximumValue = A[i + 1];
     staticMaximumValue = Math.max(staticMaximumValue, dynamicMaximumValue);
    }
   }
  }
  double LastMaximumValue = staticMaximumValue;
  return LastMaximumValue;
 }

//A METHOD FOR RETURNING MAXIMUM VALUE FROM A 1-D ARRAY OF DOUBLE VALUE ENDS
HERE


//ACTUAL COMPUTATION AND SIMULATOR IMPLEMENTATION STARTS HERE

 //A Method to run all the codes/classes made available here
 void runSimulation_actionPerformed(ActionEvent e) {

  //Re-zero relevant variables
  //Reset all relevant static variables

   flowCount = 0;
   temp_inS1=0;
   MaxAbsMatrixSolution = 0;
   timeDivision = 0;
   flowIterationNumber = 0;
   jif.setTitle("Simulation Setting");
   for (int i = 0; i < bNum; i++) {
    copyE[i] = 0;
   }
```

```
//Statement to ignore if code is still run
if (isStop == true) {

 //Set isStop false to allow thread to run. This is necessary incase
 //simulation is stopped before it completes running
 isStop = false;

 //Annonymous Thread class to handle computation on a thread
 new Thread(new Runnable() {

  //Implementation of run method of the runnable interface
  public void run() {

   //Set progress bar to indicate computation in progess
   jProgressBar1.setIndeterminate(true);
   RunNum = RunNum + 1;
   //Time at the begining of simulation in seconds
   initialTime = (System.currentTimeMillis() / 1000);

   try {
    //Passing TextFields' values to actual variables

    nwDensity = Double.parseDouble(jTextField1.getText());
    wDensity = Double.parseDouble(jTextField2.getText());
    nwVis = Double.parseDouble(jTextField3.getText());
    wVis = Double.parseDouble(jTextField4.getText());
    injTime = Double.parseDouble(jTextField5.getText());
    wInjRate = Double.parseDouble(jTextField6.getText());
    flowIterationNumber = Integer.parseInt(jTextField7.getText());
    effKnw = Double.parseDouble(jTextField8.getText());
    effKw = Double.parseDouble(jTextField9.getText());
    Swi = Double.parseDouble(jTextField10.getText());
    Sor = Double.parseDouble(jTextField11.getText());
    flowCountTolerance = Integer.parseInt(jTextField12.getText());
    flowTolerance = Double.parseDouble(jTextField13.getText());
    l = Double.parseDouble(jTextField14.getText());
    t = Double.parseDouble(jTextField15.getText());
    h = Double.parseDouble(jTextField16.getText());
    porosity = Double.parseDouble(jTextField17.getText());
    inc = Double.parseDouble(jTextField18.getText());
    Ac = Double.parseDouble(jTextField19.getText());
    KrwFitType = Double.parseDouble(jTextField20.getText());
    KroFitType = Double.parseDouble(jTextField21.getText());
    R_12a = Double.parseDouble(jTextField22.getText());
    bNum = Integer.parseInt(jTextField23.getText()) - 1;
    InitialDis = 0.0;
    Assumed_f = Double.parseDouble(jTextField24.getText());
    InitialF = Double.parseDouble(jTextField25.getText());
    epsV = Double.parseDouble(jTextField27.getText());
    inletS1 = Double.parseDouble(jTextField28.getText());
    VisCou = Double.parseDouble(jTextField29.getText()) * porosity * porosity;
    pca0 = Double.parseDouble(jTextField30.getText());
    pca1 = Double.parseDouble(jTextField31.getText());
    pca2 = Double.parseDouble(jTextField32.getText());
    pca3 = Double.parseDouble(jTextField33.getText());
    krwa0 = Double.parseDouble(jTextField34.getText());
    krwa1 = Double.parseDouble(jTextField35.getText());
    m = Double.parseDouble(jTextField36.getText());
    krwa_R2 = Double.parseDouble(jTextField37.getText());
    kroa0 = Double.parseDouble(jTextField38.getText());
    kroa1 = Double.parseDouble(jTextField39.getText());
 n = Double.parseDouble(jTextField40.getText());
 kroa_R2 = Double.parseDouble(jTextField41.getText());
 PcFitType = Double.parseDouble(jTextField44.getText());
 PrintDis = Double.parseDouble(jTextField42.getText());
 PrintTimeStep = Double.parseDouble(jTextField43.getText());

 //Compute only relevant coefficients once
 jif.setTitle("Computing coefficients");

 //Compute relevant coefficients
 calBaseCoef();
 calculateCoef(inletS1);

 //Set initial state of "FlowView" object
 f1.getContentPane().add(fv1);
 f2.getContentPane().add(fv2);
 jif.setTitle("First time step iterations");
}

catch (NumberFormatException nfe) {
 //Exception to catch non-numeric or missing values in the TextFields
 JOptionPane.showMessageDialog(null, "Missing or Non-Numeric Input Value(s).");
}

try {
 /**Solve flow and displacement equations with do-while
  * using Newton-Raphson or Newton-Jacobi Solution method
  * condition respectively
  */
 switchCF = false;
 Increamental = 0.01;
 TD:do {
  timeDivision = timeDivision + 1;
  counterG = false;
  counterL = false;
  setUnit = 1.0;

  INS:do {
   flowCount = 0;
   do {
    //Solve the flow equation using ThomasAlgorithmSolution.
    //Compute residual and iterate untill condition is met.

    flowCount = flowCount + 1; // This static variable is related to MaxAbsMatrixSolution

    //Exception (termination of method) to handle non-convergent solution
    if (flowCount > flowCountTolerance && isStop == false) {

     //Set isStop to true
     isStop = true;

     //Reset the progressbar to signify end of computation
     jProgressBar1.setIndeterminate(false);

     JOptionPane.showMessageDialog(null, "Non-Convergent Solution! Try: "+
                        "1. Increase the value of Flow Count Tolerance;"+
                        " 2.Reduce the values of Initial Assumed S* and/or MBE Check Ratio.");

     return;
    }
```

```
      //Using "ThomasAlgorithmSolution" or "newtonJacobi" method to solve tha matrix
      if (implicitNewtonRaphson.isSelected()) {
        MatrixSolution = thomasAlgorithmSolution(dFplusOne(gOfSS, xettaOfSS), dFnone(gOfSS,
xettaOfSS),
                         dFminusOne(gOfSS, xettaOfSS), Di(gOfSS, xettaOfSS));
      }
      else {
        MatrixSolution = newtonJacobi(Di(gOfSS, xettaOfSS),  dFnone(gOfSS, xettaOfSS));
      }

      //Actual flow solution
      flowSolution = flowActualSolution(MatrixSolution,timeDivision);

      //Computation of Max absolute value of MatrixSolution
      MaxAbsMatrixSolution = MaxAbsValue(MatrixSolution);

      //Make a copy of flowSolution for later re-use
      copyE = copy(flowSolution);

      //Plot fractional flow data using "FlowView" object if interactive option is selected
      if (yes.isSelected()) {
        fv1.inputdata(sat1, copyE, bNum + 2);
        fv1.plotFlowProfile();
      }
    }
    while (flowTolerance < MaxAbsMatrixSolution && isStop == false); //Do by looping

    //Copy solution (copyE) into copyETime to be used as initial condition at the next time step
    //or to be used as solution when there is convergent at the final time step.
    copyETime = copy(copyE);

    //Display current iteration status: Time step, number of flow iteration and convergence tolerance
    jif.setTitle( " Time Step = " + timeDivision + ".  Flow Count Tolerance = " + flowCount +
                ". inletS = " + inletS1 + ". epsResult = " + epsResult);

    //when variable inlet saturation is selsected:
    if (Changeable.isSelected()) {
      //Compute derivative distance with respect to saturation using
      //fractional flow equation
      dd_ds = differ_dsC(copyETime, cOfSS, gOfSS, nc, inletS1);
      //Compute distance travelled based on fractional flow equation
      normalizedDis = dis(dd_ds, deltaS1);

      //Calculate EPS (material balance check ratio)
      double sumDistance = sumDis(normalizedDis);
      epsResult = calculateEps(sumDistance);

      //check if the injected pove volume is equal to the calculated one
      if (Math.abs(epsResult) <= epsV) {
        //save the solution for late use if material balance is verified
        tempETime = copy(copyETime);
        temp_inS1 = inletS1;
        temp_deltaS1 = deltaS1;
        temp_sat1 = copy(sat1);

        tempDis = copy(normalizedDis);

        //if S* is close to 1.0 enough, make it 1.0
        if ( (1.0 - temp_inS1) <= (1.0 / (500 * (bNum + 1)))) {
          inletS2 = 1.0;
    deltaS2 = deltaS(inletS2);
    sat2 = sat(deltaS2);

    //redistribute the solution on new saturation domain for next timestep use
    interplationF = interpolateF(temp_inS1, sat2, tempETime);
    copyE = copy(interplationF);
    copyETime = copy(copyE);
    interplationX = interpolateX(temp_inS1, sat2, tempDis);
    copyDisTime = copy(interplationX);

    //recompute the mateix coefficients based the new saturation domain

    inletS1 = inletS2;
    calculateCoef(inletS1);

    //use fixed S* method for the rest of simulation
    Fixed.setSelected(true);
    Changeable.setSelected(false);
    switchCF = true;
  }
  break INS;
}

//if injected pore volume is much greater than the calculated one, increase S*
//at the normal Increamental value; if Increamental is too large, increase
//S* at decreased Increamental value

else if (epsResult > epsV) {
  counterG = true;
  if (counterL == true) {
    counterL = false;
    setUnit = setUnit * 2;
  }
  inletS2 = inletS1 + Increamental / setUnit;

  if (inletS2 > 1.0) {
    inletS2 = 1.0;

    if (timeDivision > 1) {
      deltaS2 = deltaS(inletS2);
      sat2 = sat(deltaS2);
      //redistribute the solution on new saturation domain for next timestep use
      interplationF = interpolateF(temp_inS1, sat2,tempETime);
      copyE = copy(interplationF);
      copyETime = copy(copyE);
      interplationX = interpolateX(temp_inS1, sat2, tempDis);
      copyDisTime = copy(interplationX);
    }

    //recompute the mateix coefficients based the new saturation domain
    inletS1 = inletS2;
    calculateCoef(inletS1);

    Fixed.setSelected(true);
    Changeable.setSelected(false);
    switchCF = true;

    continue INS;
  }

}
```

```
    //if injected pore volume is much less than the calculated one, decrease S*
    //at the normal Increamental value; if Increamental is too large, decrease
    //S* at decreased Increamental value
    else if (epsResult < -epsV) {
      counterL = true;
      if (counterG == true) {
        counterG = false;
        setUnit = setUnit * 2;
      }
      inletS2 = inletS1 - Increamental / setUnit;
      if (inletS2 <= 0.0) {
        inletS2 = Increamental / setUnit;
        if (inletS2 == 0.0) {
          JOptionPane.showMessageDialog(null,
             "Inlet saturation is equal to Zero, please estamite a new S* and run simulation again.");
        }
      }
    }
    //if material balance is not verified at first timestep, do not need to
    //redistribute the solution on new saturation domain;
    //if material balance is not verified at timestep other than 1,
    //redistribute the solution on new saturation domain for late use.
    if (timeDivision > 1) {
      deltaS2 = deltaS(inletS2);
      sat2 = sat(deltaS2);

      interplationF = interpolateF(temp_inS1, sat2, tempETime);
      copyE = copy(interplationF);
      copyETime = copy(copyE);
    }
    //recompute the mateix coefficients based the new saturation domain
    inletS1 = inletS2;
    calculateCoef(inletS1);
    Increamental = inletS1 - temp_inS1;
    continue INS;
  }

  //when fixed inlet saturation is selsected:
  //Compute distance travelled based on frontal advance equation
  else if (Fixed.isSelected()) {
    if (timeDivision == 1 || switchCF == true) {
      switchCF = false;
      normalizedDis = zetta(copyETime, copyETime, copyDisTime, deltaS1, timeDivision);
    }
    else {
      normalizedDis = zetta(copyETime, copyETimeTime, copyDisTime, deltaS1, timeDivision);
    }
    copyDisTime = copy(normalizedDis);
    copyETimeTime = copy(copyETime);

    break INS;
  }

}
while (Math.abs(epsResult) > epsV && isStop == false);

//Plot saturation-distance using "FlowView" object if interactive option is selected
if (yes.isSelected()) {
  if (Changeable.isSelected()) {
    fv2.inputdata(normalizedDis, temp_sat1, bNum + 2);
    fv2.plotSaturationProfile();
    }
    else {
     fv2.inputdata(normalizedDis, sat1, bNum + 2);
     fv2.plotSaturationProfile();
    }
   }

  //output simulation results at set timesteps
  if (normalizedDis[0] >= PrintDis && normalizedDis[0] <= PrintDis+0.001 ||
     timeDivision == PrintTimeStep|| timeDivision == flowIterationNumber) {

   //Calculate MBE
   double sumDistance = sumDis(normalizedDis);
   mbeResult = calculateMBE(sumDistance);

   //Time at end simulation in seconds
   finalTime = System.currentTimeMillis() / 1000;

   double injectTime =Time();

   //Using the printOutput method to print output results
   if (Changeable.isSelected()) {
    //compute potential gradient

    norPressPotenOne = potentialGradientOne(tempETime, temp_deltaS1, dd_ds);
    norPressPotenTwo = potentialGradientTwo(tempETime, temp_deltaS1, dd_ds);
    printOutput(temp_sat1, tempETime, normalizedDis, norPressPotenOne, norPressPotenTwo, injectTime,
             finalTime, initialTime, MaxAbsMatrixSolution, norFullTime, nc, ng, mr, gOfSS,
             flowIterationNumber, cOfSS, fOfSS, timeDivision, mbeResult);
   }
   else if (Fixed.isSelected()) {

    //compute the derivative of distance with respect to Saturation for
    //Fixed S* solution
    dd_ds = differ_dsF(normalizedDis, deltaS1);
    //compute potential gradient
    norPressPotenOne = potentialGradientOne(copyETime, deltaS1, dd_ds);
    norPressPotenTwo = potentialGradientTwo(copyETime, deltaS1, dd_ds);

    printOutput(sat1, copyETime, normalizedDis, norPressPotenOne, norPressPotenTwo, injectTime,
            finalTime, initialTime, MaxAbsMatrixSolution, norFullTime, nc, ng, mr, gOfSS,
            flowIterationNumber, cOfSS, fOfSS, timeDivision, mbeResult);
   }
  }
 }
 while (timeDivision < flowIterationNumber && isStop == false); //Do by looping
}
catch (NumberFormatException nfe) {

 //Exception to catch non-numeric output results
 JOptionPane.showMessageDialog(null, "Missing or Non-Numeric Ouput Value(s).",
             "Missing or Non-Numeric", JOptionPane.INFORMATION_MESSAGE);
 return;
}

//Reset the progressbar to signify end of computation
jProgressBar1.setIndeterminate(false);

if (isStop == false) {
 jif.setTitle("");
```

```
      //Message box to signify end of simulation
      JOptionPane.showMessageDialog(null, "End of Simulation Run. If No Error Message, Check Ouput File
for Results.");
     }
     //Reset isStop back to true for new run on the thesame frame
     isStop = true;

    } //run method ends here
   }).start(); //thread class ends here
  }
  else {
   //do nothing
  } //do nothing: ignore
 }

 //Save input data to file
 void saveInputData_actionPerformed(ActionEvent e) {
  File fileName;
  JFileChooser chooser = new JFileChooser("ics");
  int event = chooser.showSaveDialog(jif);
  fileName = chooser.getSelectedFile();
  if (event == JFileChooser.CANCEL_OPTION) {
   return;
  }

  try {
   //Declare variables/objects
   FileWriter FW = new FileWriter(fileName + ".dat");
   PrintWriter PW = new PrintWriter(FW, true);

   //Set coupling option
   if (viscousCoupling.isSelected()) {
    couplingSelection = 'v';
   }
   else if (capillaryCoupling.isSelected()) {
    couplingSelection = 'c';
   }
   else if (bothCouplings.isSelected()) {
    couplingSelection = 'b';
   }
   else if (noCoupling.isSelected()) {
    couplingSelection = 'n';
   }

   //Set R12 selection option
   if (r12.isSelected()) {
    r12Selection = '1';
   }
   else if (jRadioButton2.isSelected()) {
    r12Selection = '0';
   }

   //Set Inlet Saturation selection option
   Fixed.setSelected(false);
   if (Fixed.isSelected()) {
    insSelection = 'f';
   }
   else if (Changeable.isSelected()) {
    insSelection = 'c';
   }
```

```
    //Then, save through looping
    for (int i = 1; i < (44); i++) {

      //PW.println(Double.parseDouble(jTextFieldArray[countInt].getText()));
      PW.println(jTextFieldArray[i].getText());
    }

    //Save R12 selection option
    PW.println(couplingSelection);

    //Save coupling selection option
    PW.println(r12Selection);

    //Save insSelection selection option
    PW.println(insSelection);

    //Close file
    PW.close();
  }

  catch (IOException ioe) {
    //Exception to catch wrong output results.
    JOptionPane.showMessageDialog(null, "Wrong Inputs Name! Fill in a Valid Name.");
  }
}

//Open input data File
void openInputData_actionPerformed(ActionEvent e) {
  File fileName;
  JFileChooser chooser = new JFileChooser("ics");
  int event = chooser.showOpenDialog(jif);
  fileName = chooser.getSelectedFile();
  if (event == JFileChooser.CANCEL_OPTION) {
    return;
  }

  int countInt = 0;

  //Reset all values to no-value;
  for (int i = 1; i < 44; i++) {
    jTextFieldArray[countInt].setText("");
  }

  try {
    //Declare variables/objects
    FileReader FR = new FileReader(fileName);
    BufferedReader BR = new BufferedReader(FR);
    String holdText;

    //Then, read data from file to "jTextFields" through looping
    while ( (holdText = BR.readLine()) != null) {
      countInt++;

      if (countInt <= 43) {
        jTextFieldArray[countInt].setText(holdText);

        if (countInt == 23) {
          int valueOfSlide = Integer.parseInt(jTextField23.getText());
          jSlider1.setValue(valueOfSlide);
        }
  }

  //Load coupling option
  if (countInt == 44) {
   if (holdText.charAt(0) == 'v') {
    viscousCoupling.setSelected(true);
   }
   else if (holdText.charAt(0) == 'c') {
    capillaryCoupling.setSelected(true);
    jTextField29.setEnabled(false);
    jTextField29.setText("0.0");
   }
   else if (holdText.charAt(0) == 'b') {
    bothCouplings.setSelected(true);
   }
   else if (holdText.charAt(0) == 'n') {
    noCoupling.setSelected(true);
    jTextField29.setEnabled(false);
    jTextField29.setText("0.0");
   }
  }

  //Load R12 option
  if (countInt == 45) {
   if (holdText.charAt(0) == '0') {
    jRadioButton2.setSelected(true);
    jTextField22.setEnabled(false);
    r12Selection = '0';
   }
   else if (holdText.charAt(0) == 'c') {
    jRadioButton2.setSelected(true);
    jTextField22.setEnabled(true);
    r12Selection = '1';
   }
  }

  //Load Inlet Saturation option
  if (countInt == 46) {
   if (holdText.charAt(0) == 'f') {
    Fixed.setSelected(true);
    jTextField27.setEnabled(false);
    insSelection = 'f';
   }
   else if (holdText.charAt(0) == 'c') {
    Changeable.setSelected(true);
    jTextField27.setEnabled(true);
    insSelection = 'c';
   }
  }

 }

 //Close file
 BR.close();

}

catch (IOException ioe) {
 //Exception to catch wrong output results.
 JOptionPane.showMessageDialog(null, "Wrong Inputs File. Select Another.");
}
```

```
}

//A method for printing output results
void printOutput(double[] sat, double[] copyE, double normalizedDis[],double norPressPotenOne[],
            double norPressPotenTwo[],double printTime, double finalTime, double initialTime,
            double flowResidual, double normalizedTime, double nc, double ng, double mr,
            double gOfS[],int timeIntervalNumber, double cOfS[],double fOfS[],
            int flowTimestep, double mbe) {

 //Instantiating new "DecimalFormat" to be used for formating output results
 DecimalFormat dataFormatOne = new DecimalFormat("0.0000");
 DecimalFormat dataFormatTwo = new DecimalFormat("0.000000000");

 int printNum = normalizedDis.length;
 double normalizedTimee = 0;
 normalizedTimee = normalizedTime * flowTimestep;

 //Extrapolated values of distance and pressure/potential gradient at zero saturation

 try {
  //Declare variables/objects for saving output results
  FileWriter FW = new FileWriter("Output Result" + RunNum + ".dat", true);
  PrintWriter PW = new PrintWriter(FW, true);
  //Then, print(Save) Output results to file
  PW.println("--------------------------------------------------------------------------------");
  PW.println("                    Simulation Output Results                    ");
  PW.println("                      At Time=" + String.valueOf(dataFormatOne.format(printTime)) + "Seconds");
  PW.println( "--------------------------------------------------------------------------------");
  PW.print("S/No");
  PW.print('\t');
  PW.print("Norm-S");
  PW.print('\t');
  PW.print("fw");
  PW.print('\t');
  PW.print("Pot-Grad1 (Pa/Norm-X)");
  PW.print('\t');
  PW.print("Pot-Grad2 (Pa/Norm-X)");
  PW.print('\t');
  PW.print("GofS");
  PW.print('\t');
  PW.print("CofS");
  PW.print('\t');
  PW.print("FofS");
  PW.print('\t');
  PW.println("Norm-X");
  PW.print("0");
  PW.print('\t');
  PW.print(dataFormatOne.format(sat[0]));
  PW.print('\t');
  PW.print("0.000000000");
  PW.print('\t');
  PW.print(dataFormatOne.format(norPressPotenOne[0]));
  PW.print('\t');
  PW.print(dataFormatOne.format(norPressPotenTwo[0]));
  PW.print('\t');
  PW.print(String.valueOf(dataFormatTwo.format(gOfS[0])));
  PW.print('\t');
  PW.print(String.valueOf(dataFormatTwo.format(cOfS[0])));
  PW.print('\t');
  PW.print(String.valueOf(dataFormatTwo.format(fOfS[0])));
  PW.print('\t');
PW.println(dataFormatTwo.format(normalizedDis[0]));
for (int i = 1; i < printNum - 1; i++) {

  //output variables to hold numerical values
  double v1 = sat[i];
  double v2 = copyE[i - 1];
  double v4One = norPressPotenOne[i];
  double v4Two = norPressPotenTwo[i];
  double v5 = gOfS[i];
  double v6 = cOfS[i];
  double v7 = fOfS[i];
  double v3 = normalizedDis[i];

  //Convert the numeric values to String and format to 4 decimal places
  String v1Text = String.valueOf(dataFormatOne.format(v1));
  String v2Text = String.valueOf(dataFormatTwo.format(v2));
  String v4OneText = String.valueOf(dataFormatOne.format(v4One));
  String v4TwoText = String.valueOf(dataFormatOne.format(v4Two));
  String v5Text = String.valueOf(dataFormatTwo.format(v5));
  String v6Text = String.valueOf(dataFormatTwo.format(v6));
  String v7Text = String.valueOf(dataFormatTwo.format(v7));
  String v3Text = String.valueOf(dataFormatTwo.format(v3));

  //Do actual writing to file.
  PW.print(i);
  PW.print('\t');
  PW.print(v1Text);
  PW.print('\t');
  PW.print(v2Text);
  PW.print('\t');
  PW.print(v4OneText);
  PW.print('\t');
  PW.print(v4TwoText);
  PW.print('\t');
  PW.print(v5Text);
  PW.print('\t');
  PW.print(v6Text);
  PW.print('\t');
  PW.print(v7Text);
  PW.print('\t');
  PW.println(v3Text);

  //Display current iteration status: Time step, number of flow iteration and
  //convergence tolerance
  jif.setTitle(" Time Step = " + timeDivision +" Elapsed Time = " +
          dataFormatUpdateTime2.format(System.currentTimeMillis() /
          1000 - initialTime)+ " Sec." + " Saving results");
}
PW.print(printNum - 1);
PW.print('\t');
PW.print(dataFormatOne.format(sat[printNum - 1]));
PW.print('\t');
PW.print("1.000000000");
PW.print('\t');
PW.print(dataFormatOne.format(norPressPotenOne[printNum - 1]));
PW.print('\t');
PW.print(dataFormatOne.format(norPressPotenTwo[printNum - 1]));
PW.print('\t');
PW.print(String.valueOf(dataFormatTwo.format(gOfS[printNum - 1])));
PW.print('\t');
PW.print(String.valueOf(dataFormatTwo.format(cOfS[printNum - 1])));
```

```
      PW.print('\t');
      PW.print(String.valueOf(dataFormatTwo.format(fOfS[printNum - 1])));
      PW.print('\t');
      PW.println(dataFormatTwo.format(normalizedDis[printNum - 1]));

      //Write out miscellaneous data
      PW.println("------------------------------------------------------------------------");
      PW.println("Miscellaneous Computation Output Data.                  ");
      PW.println("------------------------------------------------------------------------");
      PW.print("Number of flow time step: ");
      PW.println(flowTimestep);
      PW.print("Simulation Time (sec): ");
      PW.println(dataFormatOne.format(finalTime -initialTime));
      PW.print("Total Normalized Time: ");
      PW.println(dataFormatTwo.format(normalizedTimee));
      PW.print("Normalized Time Per TimeStep: ");
      PW.println(dataFormatTwo.format(normalizedTime));
      PW.print("Discretized Saturation Domain: ");
      PW.println(dataFormatOne.format(sat[printNum - 1]));
      PW.print("Max. Normalized Distance: ");
      PW.println(String.valueOf(dataFormatOne.format(normalizedDis[0])));
      PW.print("MBE: ");
      PW.println(dataFormatOne.format(mbe));
      PW.print("Nc: ");
      PW.println(dataFormatTwo.format(nc));
      PW.print("Ng: ");
      PW.println(dataFormatTwo.format(ng));
      PW.print("Mr: ");
      PW.println(dataFormatTwo.format(mr));

      PW.println("------------------------------------------------------------------------");
      //Close file
      PW.close();
    }

    catch (IOException ioe) {
      //Exception to catch wrong output results.
      JOptionPane.showMessageDialog(null,
                                    "Wrong Outputs! Close the Currently Opened Output Files and Try Running Simulation
Again.");
    }
  }

  public void jTextField44_actionPerformed(java.awt.event.ActionEvent e) {

  }
}

class IcsClass_jTextField44_actionAdapter
    implements java.awt.event.ActionListener {
  private IcsClass adaptee;
  IcsClass_jTextField44_actionAdapter(IcsClass adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(java.awt.event.ActionEvent e) {
    adaptee.jTextField44_actionPerformed(e);
  }
}

//ACTUAL COMPUTATION AND SIMULATOR IMPLEMENTATION ENDS HERE
```

4) The source codes listings for FlowView.java

```
package ics;

import java.awt.*;
import javax.swing.*;

//This class is responsible for displaying the fractional flow profile
//and saturation profile as the compuation progresses.

public class FlowView extends JPanel {

  public int width = 210, height = 210;        // Default width and height
  public int DOT_SIZE = 2;                     // Size of a graph point
  int b_Num;                                   // Size of sat. and flow data
  double [] X = new double[b_Num];             // horizontal data
  double [] Y = new double[b_Num];             // vertical data

  //Constructor
  public FlowView() {
    setSize(width, height);
  }

  public void inputdata(double [] XX, double [] YY, int bNumm){
    b_Num = bNumm;
    X = copy(XX);  //XX converted to percentage/fraction
    Y = copy(YY);  //YX converted to percentage/fraction
  }

  public void plotFlowProfile() {
     Graphics g = this.getGraphics();
     g.setColor(getBackground());             // Clear the drawing area
     g.setClip(0, 0, width, height);
     g.fillRect(0, 0, width, height);
     g.setClip(5, 5, width - 5, height - 5);   // Reset the clip region
     g.translate(width/20, 15*height/20);         // Place origin at middle
     g.setColor(Color.black);
     drawXYAxes();                            // Draw the X and Y axes
     flowProfile();                           // Plot Fractional flow profile
   }

  public void plotSaturationProfile() {
     Graphics g = this.getGraphics();
     g.setColor(getBackground());             // Clear the drawing area
     g.setClip(0, 0, width, height);
     g.fillRect(0, 0, width, height);
     g.setClip(5, 5, width - 5, height - 5);   // Reset the clip region
     g.translate(width/20, 15*height/20);      // Place origin at middle
     g.setColor(Color.black);
     drawXYAxes();                            // Draw the X and Y axes
     saturationProfile();                     // Plot Fractional flow profile
   }

  public void drawXYAxes() {
     Graphics g = this.getGraphics();
     g.translate(width/20, 15*height/20);
     int hBound = width;                      // Use it to set the bounds
     int vBound = height;
     int tic = width /80;
```

```
      g.drawLine(0,0,hBound,0);                 // Draw X-axis
      for (int k =0; k <= hBound; k+=10)
         g.drawLine(k, 0, k, -tic);
         g.drawLine(0, 0, 0, -vBound);          // Draw Y-axis
      for (int k =0; k >= -vBound; k-=10)
         g.drawLine(0, k, +tic, k);
   }

   public void flowProfile() {
      Graphics g = this.getGraphics();
      g.translate(width/20, 15*height/20);
      int hBound = width;                         // Use it to set the bounds
      g.setColor(Color.red);
      g.fillOval((int) X[0], 0, DOT_SIZE, DOT_SIZE);              // Draw the first point

      for (int i = 1; i < b_Num-1; i++) {                 // For each pixel on x axis
        g.fillOval((int) X[i], (int) -Y[i-1], DOT_SIZE, DOT_SIZE); // Draw all other points and reverse y (for
cartesian)
      }
      g.fillOval((int) X[b_Num-1], -1, DOT_SIZE, DOT_SIZE);
   }

   public void saturationProfile() {
    Graphics g = this.getGraphics();
    g.translate(width/20, 15*height/20);
    int hBound = width ;                          // Use it to set the bounds
    g.setColor(Color.blue);

    for (int i = 0; i < b_Num; i++) {                    // For each pixel on x axis
      g.fillOval( (int) X[i], (int) - Y[i], DOT_SIZE, DOT_SIZE);  // Draw all other points and reverse y (for
cartesian)
    }
   }

   public double [] copy(double [] A){
    int j = A.length;
    double [] E = new double [j];
    for(int i = 0; i < j; i++){
     E[i] = A[i] * 100;
    }
    return E;
   }
}
```

5) The source codes listings for IcsFrame_AboutBox.java

```
package ics;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import ics.*;

// This class is to construct a frame to display the information about this software.

public class IcsFrame_AboutBox extends JDialog implements ActionListener {
 //Panels
 JPanel panel1 = new JPanel();
 JPanel panel2 = new JPanel();
JPanel insetsPanel1 = new JPanel();
JPanel insetsPanel2 = new JPanel();
JPanel insetsPanel3 = new JPanel();

//Button
JButton button1 = new JButton();

//Labels
JLabel imageLabel = new JLabel();
JLabel label1 = new JLabel();
JLabel label2 = new JLabel();
JLabel label3 = new JLabel();
JLabel label4 = new JLabel();

//Layout formats
BorderLayout borderLayout1 = new BorderLayout();

BorderLayout borderLayout2 = new BorderLayout();
FlowLayout flowLayout1 = new FlowLayout();
GridLayout gridLayout1 = new GridLayout();

//Strings
String product = "Interfacial Coupling Simulator(2.0)";
String version = "Version: 2.0";
String copyright = "Copyright (c) 2005, University of Alberta";
String comments = "Developed by Xiao Y Zhang, Based on Version 1.0 from Oluropo Rufus Ayodele";

//Constructor
public IcsFrame_AboutBox(Frame parent) {
  super(parent);
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);

  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
  pack();
}

//Component initialization
private void jbInit() throws Exception  {
  //imageLabel.setIcon(new ImageIcon(IcsFrame_AboutBox.class.getResource("[Your Image]")));
  this.setTitle("About");
  setResizable(false);
  panel1.setLayout(borderLayout1);
  panel2.setLayout(borderLayout2);
  insetsPanel1.setLayout(flowLayout1);
  insetsPanel2.setLayout(flowLayout1);
  insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
  gridLayout1.setRows(4);
  gridLayout1.setColumns(1);
  label1.setText(product);
  label2.setText(version);
  label3.setText(copyright);
  label4.setText(comments);
  insetsPanel3.setLayout(gridLayout1);
  insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
  button1.setText("Ok");
  button1.addActionListener(this);
```

```
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    this.getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
  }

  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      cancel();
    }
    super.processWindowEvent(e);
  }

  //Close the dialog
  void cancel() {
    dispose();
  }

  //Close the dialog on a button event
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button1) {
      cancel();
    }
  }
}
```

Appendix E: Derivation of Equation (6.3)

Note: interfacial coupling effects are neglected in the following derivation process.

The normalized fractional flow equation has the following form:

$$f(S,\tau) = G(S) - N_c C(S) / \frac{\partial \xi}{\partial S}. \tag{E.1}$$

When N_g=0, Equation (E.1) reduces to:

$$f(S,\tau) = F_1(S) - N_c C(S) / \frac{\partial \xi}{\partial S}. \tag{E.2}$$

Taking the derivative of Equation (E.2), one obtains:

$$\frac{\partial f}{\partial S} = \frac{dF_1}{dS} - N_c \left[\frac{\partial \xi}{\partial S} \frac{dC}{dS} - C(S) \frac{\partial^2 \xi}{\partial S^2} \right] \Big/ \left(\frac{\partial \xi}{\partial S} \right)^2. \tag{E.3}$$

Near S=0, $\lim_{S \to 0} \mathrm{C(S)} = 0$. Hence, one obtains:

$$\lim_{S \to 0} \frac{\partial f}{\partial S} = \frac{dF_1}{dS} - N_c \frac{dC}{dS} \frac{\partial S}{\partial \xi}. \tag{E.4}$$

$$F_1(S) = \frac{M_r k_{r1}(S)}{M_r k_{r1}(S) + k_{r2}(S)}, \tag{E.5}$$

$$\frac{dF_1}{dS} = \frac{(M_r k_{r1} + k_{r2}) M_r k'_{r1} - M_r k_{r1} (M_r k'_{r1} + k'_{r2})}{(M_r k_{r1} + k_{r2})^2}, \tag{E.6}$$

Near S=0, $\lim_{S \to 0} k_{r1} = 0$, and $\lim_{S \to 0} k_{r2} = 1$.

$$\lim_{S \to 0} \frac{dF_1}{dS} = M_r \frac{dk_{r1}}{dS}, \tag{E.7}$$

$$C(S) = -\frac{1}{M_r} F_1(S) k_{r2}(S) \frac{d\pi_c}{dS}, \tag{E.8}$$

$$\frac{dC}{dS} = -\frac{1}{M_r}\left\{F_1 k_{r2} \frac{d^2\pi_c}{dS^2} + \frac{d\pi_c}{dS}\left[F_1 k'_{r2} + k_{r2}\frac{dF_1}{dS}\right]\right\}, \tag{E.9}$$

Near S=0, $\lim\limits_{S\to 0} F_1 = 0$, and $\lim\limits_{S\to 0} k_{r2} = 1$.

$$\lim_{S\to 0}\frac{dC}{dS} = -\frac{1}{M_r}\frac{d\pi_c}{dS}\frac{dF_1}{dS}, \tag{E.10}$$

Substituting Equation (E.7) into Equation (E.10) yields:

$$\lim_{S\to 0}\frac{dC}{dS} = -\frac{d\pi_c}{dS}\frac{dk_{r1}}{dS}, \tag{E.11}$$

Substituting Equations (E.7) and (E.11) into Equation (E.4), one obtains:

$$\lim_{S\to 0}\frac{\partial f}{\partial S} = M_r\frac{dk_{r1}}{dS} + N_c\frac{d\pi_c}{dS}\frac{dk_{r1}}{dS}\frac{\partial S}{\partial \xi}. \tag{E.12}$$