Numerical Simulations of Electrically Induced Chloride Ion Transport and Moisture Permeability through Cracked Concrete

by

Pu Yang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2014 by the
Graduate Supervisory Committee:

Narayanan Neithalath, Chair
Subramaniam Dharmarajan
Barzin Mobasher

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

The main objective of this study is to numerically investigate: (i) the ionic transport, especially chloride ion penetration into cementitious materials under imposed electric fields, and (ii) moisture transport through cracked concretes as a function of the crack geometry.

Numerical methods were implemented to simulate the ionic transport process, based on coupling the Nernst-Planck equation and Poisson's equation to account for transport dominated by electromigration. This mathematical model was also modified to account for the chloride binding mechanism (physical and chemical trapping of chlorides by the cement hydrates) and the concentration dependence of the diffusion coefficient of each ion in the transport process. To validate the numerical model, experimental data from a companion work was used in this study. The non-steady state migration test, which is one of the common accelerated chloride ion transport test, is numerically simulated. The simulation provides a linear relationship between ionic concentration and ionic flux, which indicates that the diffusion part is negligible under a strong external voltage environment. The numerical models along with adjustments for the concentration-dependent diffusion coefficients, a pore structure factor (from electrical measurements) and chloride binding considerations are found to be successful in predicting the chloride penetration depth into plain and modified concretes under imposed electrical potentials.

Moisture transport through cracked concrete was examined in the second part of this thesis. To better understand the crack's influence on the permeability, modified Louis' equation was chosen to relate the permeability with crack characteritsics. 3D concrete crack models were developed using a MATLAB program with distinct crack tortuosities, roughnesses and sizes. As a comparison, Navier-Stokes equation and the Lattice

Boltzmann method were also applied on the 3D model of the cracked concrete to evaluate their permeability. The methodology developed here is expected to be useful in understanding the influence of cracking on moisture transport, and when properly coupled with an ionic transport model that will be further developed, helps comprehensively understand the coupling effects of moisture and ionic transport on deterioration in concrete structures.

To my grandmother, may god be with you

To my families and friends who always understand and support me

ACKNOWLEDGEMENTS

I would like to specially thank my advisor Dr. Narayanan Neithalath for guiding me in my research. I also want to extend my appreciation to Dr. Subramaniam D. Rajan and Dr. Barzin Mobasher, who served as my committee members, helping and supervising my progress in Master's degree program.

I also would like to thank Ben Rehder, who taught me many skills when I started my work. I also want to thank Sumanta Das, who did a great job in helping me with the paperwork of my thesis.

I also would like to express my gratitude to my dear colleagues and friends, Akash Dakhane, Kirk Vance, Matthew Aguayo, Aashay Arora and Sateesh Madavarapu for their help and support.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

Table Page

PART I. NUMERICAL SIMULATION OF ACCELERATED CHLORIDE ION TRANSPORT INTO SATURATED CONCRETE

1 INTRODUCTION

1.1 Introduction to Ionic Transport Phenomenon

1.1.1 Free Diffusion

Diffusion is one kind of transport phenomena (unlike convection or advection) which leads to a movement of particles from an area of high concentration to an area of low concentration, resulting in a uniform distribution of the substance in the medium (Fig. 1.1).

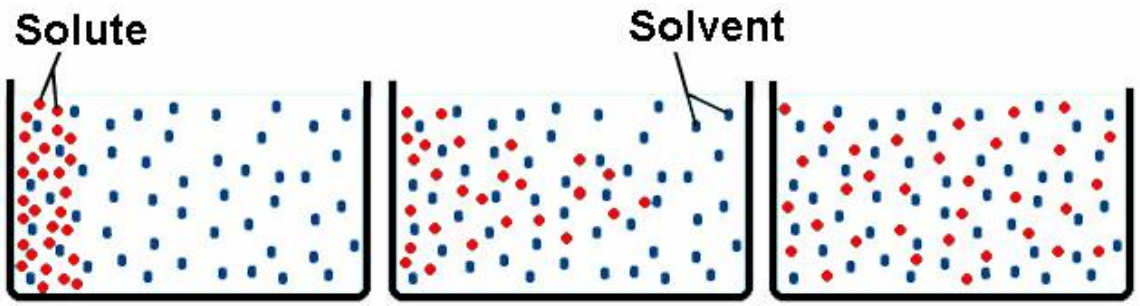


Figure 1.1. Schematic diagram of diffusion process (1)

As shown in Fig 1-2, if the concentration in Region 1 and Region 2 are C_1 and C_2 (mol/L) respectively and the distance between them is dx, then the flux from region 1 to region 2 is given as:

$$J = D\frac{C_1 - C_2}{\Delta x} = -D\frac{dC}{dx} \tag{1-1}$$

Where D is known as the diffusion coefficient (m^2/s), and is a property of material.

Eq. (1-1) is known as Fick's 1st law and is applicable to only steady-state diffusion since concentration values are independent with time.

When the concentration is variable with time,

$$\frac{C_{x,t+\Delta t} - C_{x,t}}{\Delta t} = \frac{\partial J}{\partial x} = \frac{J_x - J_{x+\Delta x}}{\Delta x} \tag{1-2}$$

where:

$$J_{x+\Delta x} = J_x + \frac{\partial J_x}{\partial x} dx \tag{1-3}$$

Substituting Eq. (1-3) to Eq. (1-2):

$$\frac{\partial C}{\partial t} = -\frac{\partial J_x}{\partial x} = -\frac{\partial \left(-D\frac{\partial C}{\partial x}\right)}{\partial x} = D\frac{\partial^2 C}{\partial x^2} \tag{1-4}$$

This is known as Fick's 2nd law, which is the basic representation for non-steady-state diffusion of any ionic species into another medium.

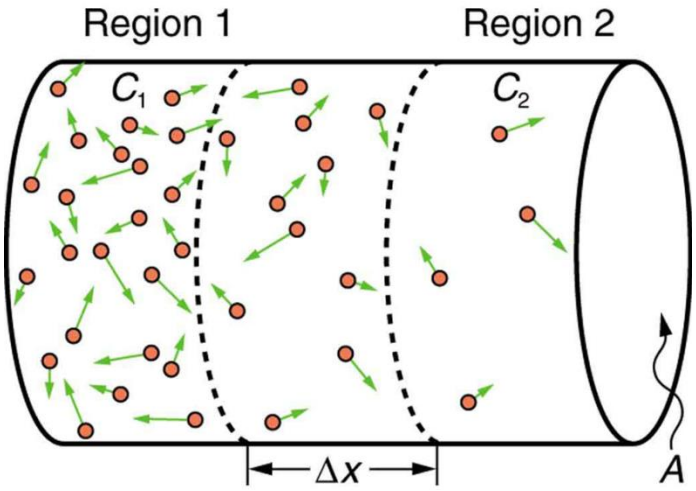


Figure 1.2. Schematic diagram of diffusion process (2)

1.1.2 Chloride Ion Diffusion into Saturated Concrete and Implications

Chloride-induced corrosion is the most important factor that affects the durability of reinforced concrete structures. Once the chloride concentration value near the reinforcing steel reaches a certain level, corrosion begins. Every year, enormous economic loss is caused by corrosion and huge sums of money is spent to repair corrosion – damaged structures.

During the process of diffusion, a certain fraction of chloride ions are retained by the hydration products of the binder in concrete, either through chemical binding or physical adsorption**[1]**, which delays the diffusion. The total chloride ions in the hardened hydrated cement paste in concretes can be divided into two types: free chloride ions, existing in the pore solution as mobile ions and contribute to further corrosion of steel and bound chloride ions, which are attached to various hydration products **[1]**. Relationships between the total, free and bound chloride ions in concrete are very important for the development of models for service life prediction of reinforced concrete structures with respect to reinforcement corrosion **[2]**. Freundlich adsorption isotherm and Langmuir adsorption isotherm are two main isotherms which describe the relationship between free and bound chlorides. They are given as:

$$C_b = k_b C_f^n \tag{1-5}$$

$$C_b = \frac{\alpha C_f}{1 + \beta C_f} \tag{1-6}$$

In which C_b is the binding chloride concentration over the sample and C_f is the free chloride concentration in pore solution. k_b, n, α, and β are binding constants varying with concrete binder compositions. Eq. (1-5) corresponds to Freundlich isotherm and Eq. (1-6) refers to Langmuir isotherm. Tang [**3**] has showed that Freundlich isotherm can describe chloride binding in free-chloride concentration ranging from 0.01 – 1 mol/L while Langmuir isotherm is appropriate for low free-chloride concentration (<0.05 mol/L). Zabara [**4**] even showed that for a higher chloride concentration, Freundlich isotherm also describes the binding correctly.

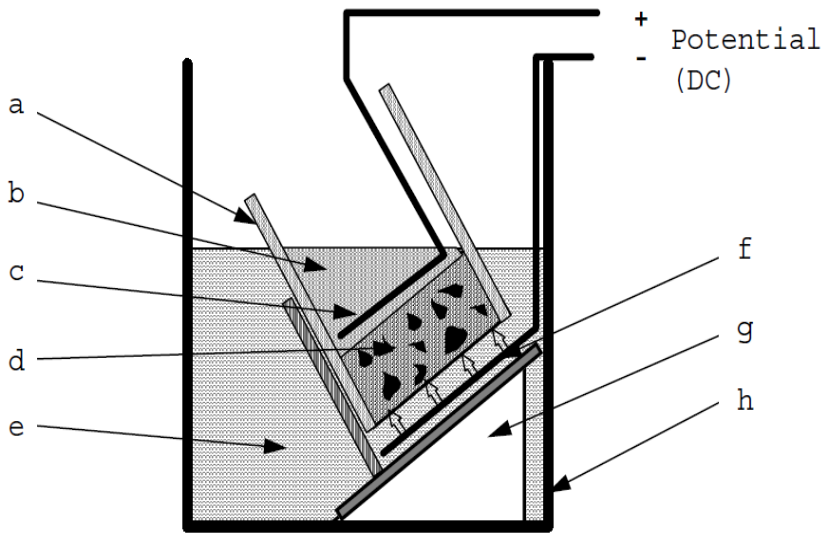
In saturated concrete, the movements of ions only take place in the liquid phase that occupies a fraction of the total porous volume. When different ionic species move in

same solution, their velocities vary due to the changes in diffusion coefficients. Since ions are charged particles, an electrical field is created, which is known as membrane potential. It can slow down the faster ions and accelerate the slower ones **[5]**.

1.2 Non-Steady State Migration Test

Several tests have been developed to understand the accelerated transport of chlorides through concrete. Among them the non-steady state migration test and the rapid chloride permeability test are the more common ones. Since the RCP test is reported to have several drawbacks, the predominant one being the joule heating associated with the large voltages used in the test [**6**], non-steady state migration test is regarded to be a more appropriate one to indicate the transport resistance of chlorides to ionic movement.

Non-steady state migration test (NSSM) is generally carried out in accordance with NT Build 492 **[7]**, on 50 mm thick concrete discs, which are preconditioned by vacuum saturating with calcium hydroxide solution, after the respective curing durations. 2N NaCl and 0.3 N NaOH solutions are used as catholyte and anolyte solutions. An initial voltage of 30 V is applied, and initial current is recorded. The applied voltage and test duration are chosen based on the initial current. The test setup is shown in Fig. 1.3.



a. Rubber sleeve
b. Anolyte
c. Anode
d. Specimen
e. Catholyte
f. Cathode
g. Plastic support
h. Plastic box

Figure 1.3. NT Build 492 test setup **[7]**

The non-steady state migration coefficient is given as:

$$D_{nssm} = \frac{RT}{zFE}\frac{x_d - \alpha\sqrt{x_d}}{t} \tag{1-7}$$

$$\alpha = 2\sqrt{\frac{RT}{zFE}}erf^{-1}\left(1 - \frac{2c_d}{c_0}\right) \tag{1-8}$$

$$E = \frac{U - 2}{L} \tag{1-9}$$

In which U is the absolute voltage in V, L is the specimen thickness in m, z is the valence of the chloride ion, F is the Faraday constant, R is the molar gas constant, T is the average value of initial and final temperatures in K, x_d is the average value of the penetration depth in m, t is the test duration in s, c_d is the chloride concentration at which

silver nitrate changes to silver chloride (0.07 N), and c_0 is the chloride concentration of the catholyte solution (2 N).

1.3 Experimental Program used in a Companion Study to Validate the Numerical Models

The materials used in this study are: a commercial Type I/II ordinary portland cement (OPC) conforming to ASTM C 150, Class F fly ash and metakaolin conforming to ASTM C 618, and limestone powder conforming to ASTM C 568. The chemical compositions of these materials are listed in Table 1.1. The concrete mixtures were proportioned by replacing 20% or 35% of the OPC by (by volume) by limestone or fly ash in the binary blends, or combination of limestone and fly ash or limestone and metakaolin in the ternary blends. More details can be found in [**8**].

Table 1.1. Chemical composition of the component materials

Component (%)	OPC	Fly ash	Metakaolin
SiO_2	21.0	58.4	51.7
Al_2O_3	3.61	23.8	43.2
Fe_2O_3	3.47	4.19	0.5
CaO	63.0	7.32	--
MgO	3.26	1.11	--
SO_3	3.04	0.44	--
Na_2O	0.16	1.43	--
K_2O	0.36	1.02	--
LOI	2.13	0.50	0.16

All the concretes were proportioned with a water-to-powder ratio of 0.40 by mass. In addition to the control OPC concrete, the mixtures included binary mixtures of limestone or fly ash at cement replacement levels of 20 and 35%, and ternary mixtures with either 10% or 25% of fly ash or 10% of metakaolin with limestone being added to achieve the total 20% or 35% OPC replacement volume (see Table 1.2). The replacement level of

OPC with metakaolin was limited to 10% so as to avoid drastic loss of workability. The aggregate was 70% by volume over the concrete. All the concrete specimens were stored in a moist chamber for RH > 97% and at a temperature of 23 ± 2^{o}C until the desired age of testing. The non-steady state migration test described in Section 1.2 was performed on all the specimens after 28 and 56 days of hydration.

Table 1.2. Mixture proportions used in this study for 1 m^3 of mortar or concrete

Mixture	Cement (kg)	Limestone (kg)	Fly ash (kg)	Metakaolin (kg)	Coarse agg. (kg)	Fine agg. (kg)
Plain	480	0	0	0	1066	661
LS 20	395	85	0	0	1065	660
FA 20	399	0	80	0	1065	660
LS 10 MK 10	398	43	0	38	1063	659
LS 35	327	151	0	0	1061	658
FA 35	333	0	145	0	1061	658
LS 25 MK 10	330	109	0	39	1060	657

The chloride penetration depths from the migration test are shown in Table 1.3. The concrete with 35% cement replacement by limestone gives the largest penetration depth while the concrete containing 10% limestone + 10% metakaolin replacement case has a minimum penetration process.

Table 1.3. NSSM test results (chloride penetration depth)

Mixture	Plain	LS 20	LS 10 MK 10	LS 35	LS 25 MK 10	FA 20	FA 35
Depth (mm)	25.37	30.71	14.10	32.50	19.75	20.75	19.50

2 MATHEMATICAL MODELS TO SIMULATE NON-STEADY STATE CHLORIDE MIGRATION INTO CONCRETE

2.1 Basic Ionic Transport Equation

As discussed above, transport of chloride into saturated porous concrete is a complex process. Different parameters influence the process in different ways. Fick's 2nd law is no longer suitable for describing ionic movement in concrete subjected to an electric field. In this thesis, extended Nernst-Planck equation is chosen to describe the process:

$$J_i = -D_i\left(\frac{\partial C_i}{\partial x} + \frac{z_i F}{RT} C_i \frac{\partial \Psi}{\partial x} + C_i \frac{\partial \ln \gamma_i}{\partial x} - C_i V_x\right) \quad (2\text{-}1)$$

Here J_i is the flux value of ionic species i, expressed as mole per square meter second (mol/m^{2}s); C_i is the ionic concentration in pore solution, expressed as mole per cubic meter (mol/m^3); D_i is the effective diffusion coefficient, expressed as square meter per second (m^2/s); z_i is the valence number; F is the Faraday constant (9.648534×10^4 C/mol); R is the ideal gas constant (8.3145 J/(mol·K)); T is the absolute temperature (K); Ψ is the electrical potential (V); γ_i is the chemical activity coefficient and V_x is the bulk velocity of the fluid.

There are four parts in the right-hand side of the equation, each of them corresponds to a different mechanism. The first part describes the ionic movement driven by the concentration gradient. The second part represents the effect of electrical field, which may be a result from the co-action of membrane potential and external electrical field. However, when the external electrical field is strong enough [i.e. in a migration test], the membrane potential can be negligible. The third part is due to the chemical activity. Since most of test results show that the influence from chemical activity gradient is negligible **[9,10]**, this part can be omitted. The confirmation for this is shown in this thesis also, in a

later section (Sec. 3.4). The fourth part is the advection term. When no pressure gradient exists, this term can also be omitted. **[5]**

As a result, Eq. (1-5) becomes:

$$J_i = -D_i\left(\frac{\partial C_i}{\partial x} + \frac{z_i F}{RT} C_i \frac{\partial \Psi}{\partial x}\right) \tag{2-2}$$

In this thesis, Na^+, K^+, Cl^- and OH^- ions are considered as the main ions taking part in the ions transport process. The Na^+, K^+, and OH^- ions are the dominant ions in a cement paste pore solution while the Cl^- ions are induced into the system during the NSSM test.

The transport of ionic species in the concrete can be described by the mass conservation of individual ionic species as follows:

$$\frac{\partial (pC_i)}{\partial t} = D_i \frac{\partial}{\partial x}\left(\frac{\partial C_i}{\partial x} + \frac{z_i F}{RT} C_i \frac{\partial \Psi}{\partial x}\right) \tag{2-3}$$

Where p is the porosity of concrete.

Poisson's equation is needed for evaluating the electrical potential $\Psi(x,t)$ because the assumption that the electric potential does not vary with specimen depth is invalid.

$$\nabla^2 \Psi = -\frac{F}{\varepsilon_0 \varepsilon_r} \sum_i C_i z_i \tag{2-4}$$

Eq. (2-4) is the dimensionless form of Poisson equation. Where $\varepsilon_0 = 8.854 \times 10^{-12} C/(V \cdot m)$ is the permittivity of a vacuum. $\varepsilon_r = 78.3$ is the relative permittivity of water at a temperature of 25°C.

Solving Eq. (2-3) and Eq. (2-4) gives the simulation results of the ionic transport process. Finite element method or finite difference method are required to solve these equations.

2.2 Numerical Process to Solve System Equations

Finite element method is one of the most powerful methods to solve engineering problems. By discretizing – the process of splitting the problem domain into elements, complicated problems can be separated into small elements with certain degrees of freedom, which makes the solving process much easier. In this thesis, both finite element (FE) and finite difference (FD) method are applied to solve all the partial differential equations. In this section, detailed deviations have been presented.

A full version of all the system equations is:

$$p\frac{\partial C_i}{\partial t} = \frac{\partial}{\partial x}\left(D_i\frac{\partial C_i}{\partial x} + D_i z_i \frac{F}{RT} C_i \frac{\partial \Psi(x,t)}{\partial x}\right) \tag{2-5}$$

$$\nabla^2 \Psi = -\frac{F}{\varepsilon_0 \varepsilon_r}\sum_i C_i z_i \tag{2-6}$$

A two-node linear element has been used during the solution process. Uniform mesh is applied to the problem. Each element has the same length of *L/n*, where *L* is the thickness of the sample and *n* is the number of elements.

The problem domain is:

$$0 \le x \le L, t \ge t_0 \tag{2-7}$$

Where L is the thickness of the sample and t_0 is set to 0.

Only Dirichlet boundary condition is applied to this problem:

$$\text{For } t \ge t_0, \begin{cases} C_i(x_0,t) = C_i^0 \\ C_i(x_L,t) = C_i^L \end{cases} \tag{2-8}$$

Where C_i^0 and C_i^L are the concentration values in external solutions for i-th species, which are considered as constants during the test. This is because a non- steady state

migration test has highly concentrated ionic solutions as the catholyte and anolyte and their concentrations do not change appreciably with time.

The initial conditions are:

$$\text{At } t_0(0 \le x \le L),\ C_i(x, t_0) = C_i^x \tag{2-9}$$

In which C_i^x is the concentration value in pore solution for i-th species at each node.

The element equations have been generated by Galerkin's method. The trial solution of the system can be assumed as:

$$c_i(x,t) = \sum_{j=1}^{2} \phi_j(x) a_j^i(t) \tag{2-10}$$

In which $c_i(x,t)$ is the concentration value of the i-th species, $\phi_j(x)$ is the shape function and $a_j^i(t)$ is the time-dependent concentration value at j-th node of i-th species.

For two-node linear element, the shape functions are:

$$\begin{cases} \varphi_1(x) = \dfrac{x - x_2}{x_1 - x_2} \\ \varphi_2(x) = \dfrac{x - x_1}{x_2 - x_1} \end{cases} \tag{2-11}$$

The elementary equation of the system is:

$$\boldsymbol{C}\left\{\frac{da(t)}{dt}\right\} + \boldsymbol{K}\{a(t)\} = \{F(t)\} \tag{2-12}$$

Where $a(t)$ is the unknown variable vector on each element in terms of their nodal values at each time-step:

$$a(t) = \{c_{11} \quad c_{21} \quad c_{31} \quad c_{41} \quad V_1 \quad c_{12} \quad c_{22} \quad c_{32} \quad c_{42} \quad V_2\}^T \tag{2-13}$$

In which the subscripts *i* and *j* in c_{ij} stand for the ion *i* at node *j* in a given element.

$$\boldsymbol{C} = \int \{\boldsymbol{N}\}^T \{\boldsymbol{A_1}\}\{\boldsymbol{N}\}\, dx \tag{2-14}$$

$$\boldsymbol{K} = \int (\{\boldsymbol{w}\}\{\boldsymbol{A_2}\}\{\boldsymbol{n}\} + \{\boldsymbol{W}\}\{\boldsymbol{A_3}\}\{\boldsymbol{N}\} + \{\boldsymbol{w}\}\{\boldsymbol{A_4}\}\{\boldsymbol{n}\})\, dx \tag{2-15}$$

In which $\boldsymbol{N}$ is shape function matrix and $\boldsymbol{n}$ is it's first order derivative, $\boldsymbol{W}$ is the weighting matrix and $\boldsymbol{w}$ is it's first order derivative. Here they can be expressed as:

$$\boldsymbol{N} = \begin{bmatrix} \varphi_1 & & & & & \varphi_2 & & & & \\ & \varphi_1 & & & & & \varphi_2 & & & \\ & & \varphi_1 & & & & & \varphi_2 & & \\ & & & \varphi_1 & & & & & \varphi_2 & \\ & & & & \varphi_1 & & & & & \varphi_2 \end{bmatrix} \tag{2-16}$$

$$\boldsymbol{n} = \begin{bmatrix} \varphi_{1x} & & & & & \varphi_{2x} & & & & \\ & \varphi_{1x} & & & & & \varphi_{2x} & & & \\ & & \varphi_{1x} & & & & & \varphi_{2x} & & \\ & & & \varphi_{1x} & & & & & \varphi_{2x} & \\ & & & & \varphi_{1x} & & & & & \varphi_{2x} \end{bmatrix} \tag{2-17}$$

$$\boldsymbol{W} = \boldsymbol{N}^T \tag{2-18}$$

$$\boldsymbol{w} = \boldsymbol{n}^T \tag{2-19}$$

In eq. (2-15) there are three matrices $\boldsymbol{A_2}$, $\boldsymbol{A_3}$ and $\boldsymbol{A_4}$, which represent three different mechanisms: $\boldsymbol{A_2}$ represents on the diffusion process, $\boldsymbol{A_3}$ couples the ionic concentration together with the electrical potential (Poisson's equation) and $\boldsymbol{A_4}$ considers the electrical migration term [**11**,**12**]. $\boldsymbol{A_1}$ is the basic capacity matrix including the porosity.Matrixes from $\boldsymbol{A_1}$ to $\boldsymbol{A_4}$ can be expressed as:

$$\boldsymbol{A_1} = p \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 0 \end{bmatrix} \tag{2-20}$$

$$\boldsymbol{A_2} = \begin{bmatrix} D_1 & & & & \\ & D_2 & & & \\ & & D_3 & & \\ & & & D_4 & \\ & & & & 1 \end{bmatrix} \tag{2-21}$$

$$\boldsymbol{A_3} = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ -z_1 & -z_2 & -z_3 & -z_4 & 0 \end{bmatrix} \tag{2-22}$$

$$\boldsymbol{A_4} = \begin{bmatrix} \frac{D_1 z_1 F}{RT} c_1 \\ \frac{D_2 z_2 F}{RT} c_2 \\ \frac{D_3 z_3 F}{RT} c_3 \\ \frac{D_4 z_4 F}{RT} c_4 \\ 0 \end{bmatrix} \tag{2-23}$$

The time discretization is performed using an explicit Euler scheme,

$$\boldsymbol{C}\left\{\frac{a_{n+1} - a_n}{\Delta t}\right\} + \boldsymbol{K} a_n = F_n \tag{2-24}$$

Defining the matrices

$$\begin{aligned} \overline{\boldsymbol{K}} &= \boldsymbol{C} + \Delta t \boldsymbol{K} \\ \overline{\boldsymbol{F}} &= \boldsymbol{C}\{a_n\} \end{aligned} \tag{2-25}$$

The system equation can then be rewritten as

$$\overline{\boldsymbol{K}}\{a_{n+1}\} = \overline{\boldsymbol{F}} \tag{2-26}$$

This system of equations can be solved by Gauss-Elimination method.

A central finite difference approximation is applied to obtain the flux value based on Eq. (2-2);

$$\frac{\partial C_i^j}{\partial x} = \frac{C_i^{j+1} - C_i^{j-1}}{2\Delta x} \tag{2-27}$$

2.3 Optimization of the Solution

Since the governing equation is a convection—diffusion equation, the accuracy of the solution is heavily dependent on the Peclet number, which is defined as:

$$\boldsymbol{Pe} = \frac{U}{2}\frac{h}{D} \tag{2-28}$$

Where h is the length of element and $U = \left| D\frac{F}{RT}\frac{\partial \Psi}{\partial x} \right|$ is the migration term.

Any values of Peclet number larger than 1 will cause heavy oscillation and give unstable solutions. To optimize the calculation method and reduce the number of elements needed, a Petrov-Garlerkin method is induced [**13**].

The weighting matrix can be modified as:

$$\boldsymbol{W_i} = \boldsymbol{N_i} + \alpha_{opt}\frac{h}{2}\frac{d\boldsymbol{N}_i}{dx}(sign\ U) \tag{2-29}$$

Where

$$\alpha_{opt} = cosh|P_e| - \frac{1}{|P_e|} \tag{2-30}$$

The sign is depend on whether U towards or away from the node.

The current density carried by all the four species passing through the specimen can be calculated by the following equation:

$$I = F\sum_{k=1}^{4} z_k J_k \tag{2-31}$$

In which I is current density, expressed in A/m^2.

The current at each time step is calculated according to the following equation:

$$I_c = S \cdot I \tag{2-32}$$

In which S is the surface area of the sample.

3 NUMERICAL SIMULATION AND MODEL MODIFICATION

3.1 Determination of Ionic Concentration in Pore Solution

Multiple factors such as the binder proportions, degree of hydration and the chemical composition of the ingredients can affect the initial ionic concentration of the pore solution concrete. In this study, pore solution composition is determined using Taylor's model. The degrees of hydration of the binders in the concrete at different ages were obtained from [**8**].

Taylor [**14**] proposed a model for predicting alkali ion concentrations in the pore solution of hydrated cement paste with the total amount of Na_2O and K_2O in the components and the water available for the pore solution. The alkali ions exist both in the pore solution and the hydration products, and the amount of ions occupied by hydration products is assumed proportional to its concentration in the solution and the amount of products as adsorbent [**15**]. Thus, the concentration of ions in the solution can be calculated from the remaining amount of ions and the volume of solution. Experimental equations are applied to estimate the amounts of alkali ions released by hydration process and the hydration products. The pore solution volume is computed from the total water content in the paste and products. A web interface developed by National Institute of Standards and Technology (NIST) can do the calculation based on the input information provided by users.

The pore solution composition of mixtures estimated based on the mixtures proportions can be found in Table 1.2. The initial ionic concentrations of all the mixtures are listed in Table 3.1.

Table 3.1 Initial ionic concentration in pore solution (mol/L)

Mixtures	K^+	Na^+	OH^-	Cl^-
OPC	0.21	0.14	0.35	0.00
LS20	0.16	0.11	0.27	0.00
LS10 MK10	0.16	0.11	0.27	0.00
LS35	0.12	0.08	0.20	0.00
LS25 MK10	0.13	0.09	0.22	0.00
FA20	0.25	0.30	0.55	0.00
FA35	0.27	0.40	0.67	0.00

3.2 Effective Ionic Diffusion Coefficients

Diffusion coefficients of ions determine the ionic diffusing speed in media. The media environment can drastic influence the effective ionic diffusion coefficients. In this section, a series of work has been done to find out the effective ionic diffusion coefficients for ions transportation in saturated concrete.

3.2.1 Pore Structure Dependent Effective Diffusion Coefficient

In many past works, researchers have used several means to define the ionic diffusion coefficients in predictive models. In this study, the diffusion coefficient D_i is defined as:

$$D_i = p\beta D_i^{inf} \tag{3-1}$$

In which β is the pore connectivity of the material and p is the porosity of the material. D_i^{inf} is the diffusion coefficient of the *i*-th species in infinite dilution, the values of which can be found in Table 3-2.

Table 3.2 Diffusion Coefficients of Species in Infinite Dilution **[16]**

Species	Diffusion Coefficient (10^{-9} m^2/s)
Na^+	1.334
K^+	1.957
OH^-	5.273
Cl^-	2.032
Ca^{2+}	0.792
SO_4^{2-}	1.065

The bulk specimen conductivity is related to the pore solution conductivity (σ_0) and a pore structure factor, defined herein as the product of porosity (p) and pore connectivity (β) [**17**,**18**,**19**]) as:

$$\sigma_{eff} = p\beta\sigma_0 \tag{3-2}$$

The conductivity of the pore solution was predicted by the procedure developed by Snyder [**20**]:

$$\sigma_0 = \sum z_i C_i \lambda_i \tag{3-3}$$

Where z_i is the species valence, C_i is the ionic concentration and λ_i is the equivalent conductivity which can be expressed as:

$$\lambda_i = \frac{\lambda_i^0}{1 + G_i I_M^{1/2}} \tag{3-4}$$

The values of the equivalent conductivity (λ_i^0) of ionic species in infinite dilution and the conductivity coefficient (G_i) are given in Table 3-3. The ionic strength I_M is given as:

$$I_M = \frac{1}{2}\sum z_i^2 C_i \tag{3-5}$$

Table 3.3 Equivalent conductivity at infinite dilution and conductivity coefficients for Na^+, K^+ and OH^- at 25℃ [**20**]

Species	λ^0 (cm^2S/mol)	G $(mol/L)^{-1/2}$
Na^+	50.1	0.733
K^+	73.5	0.548
OH^-	198.0	0.353

Thus, the pore connectivity can be calculated as:

$$\beta = \frac{\sigma_{eff}}{p\sigma_0} \tag{3-6}$$

A companion study [**8**] has obtained the porosity of the specimens from mercury intrusion porosimetry and effective connectivity from electrical impedance spectroscopy using the measured bulk electrical resistance.

Both the porosity and pore connectivity for the chosen concretes are shown in Table 3.4.

Table 3.4 Porosity and pore connectivity (dimensionless) of different concretes

Mixture	Plain	LS 20	LS 10 MK 10	LS 35	LS 25 MK 10	FA 20	FA 35
Porosity	0.099	0.122	0.104	0.132	0.129	0.127	0.132
Pore connectivity	0.028	0.030	0.010	0.040	0.014	0.014	0.009

Fig. 3.1 shows the predicted chloride concentration profile at the end of NSSM test for all the seven mixes. It can be noticed from this figure that, when the diffusion coefficient mentioned above is used, after 24 hours of NSSM test simulations, the chloride ions are seen to have penetrated the entire depth of the sample (50 mm). This is contradictory to the experimental test results where the largest penetration depth was only 33 mm. A possible reason for such an error could be from the effective diffusion coefficients of the migrating species used in the simulations, which is taken to be the self-diffusion coefficients of the ions multiplied by the product of porosity and pore connectivity (Eq. (3-1)). Such a definition results in an over-prediction of depth-dependent chloride profiles and the penetration depth.

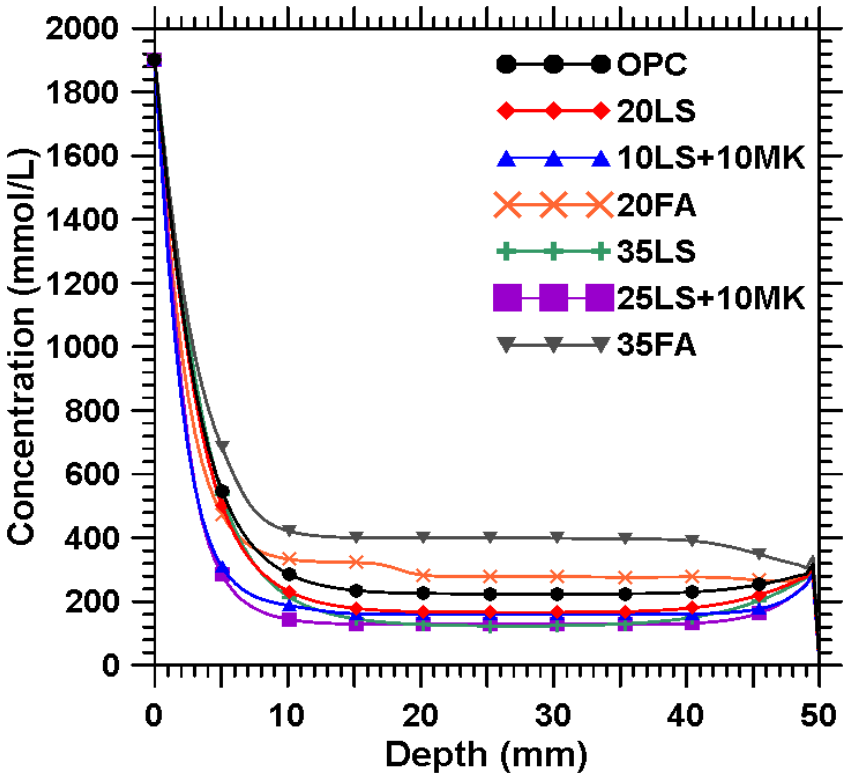


Figure 3.1. Chloride concentration profile for original definition of D

3.2.2 "Retarded" Effective Diffusion Coefficient

To find out the influence of lowering the diffusion coefficients on the overall penetration depth, the effective diffusion coefficients (D_i) were "retarded". Based on the previous definition, Eq. (3-1) was modified by multiplying the right hand side by a retardation factor ranging between 0.125-to-0.200 and the simulations were repeated.

The results of this new series of simulation with depressed diffusion coefficients in terms of chloride concentration profiles are shown in Fig. 3.2. Compared with the experimental results, the predicted penetration process is effectively retarded and an expected trend is obtained. The rationale for reducing the effective diffusion coefficients may lie in the fact that effective ionic diffusion coefficients of ions in a sample undergoing migration is not the same at all locations and are concentration dependent. It has also been reported that the apparent chloride migration coefficient may vary as a function of spatial position

during the accelerated migration test due to chloride binding and non-equilibrium conditions, i.e., equilibrium between the free and bound Cl^- ions is not achieved during NSSM tests [**21**]. The local binding capacity and the local migration coefficient depend on the local concentrations of free and bound chlorides. Thus, the low concentration of chlorides at the penetration front helps progress the front with little binding, but as the local chloride concentration increases behind the front, so does the binding capacity [**21**]. These factors are then expected to result in concentration dependent ion transport functions to adequately represent the ionic movements.

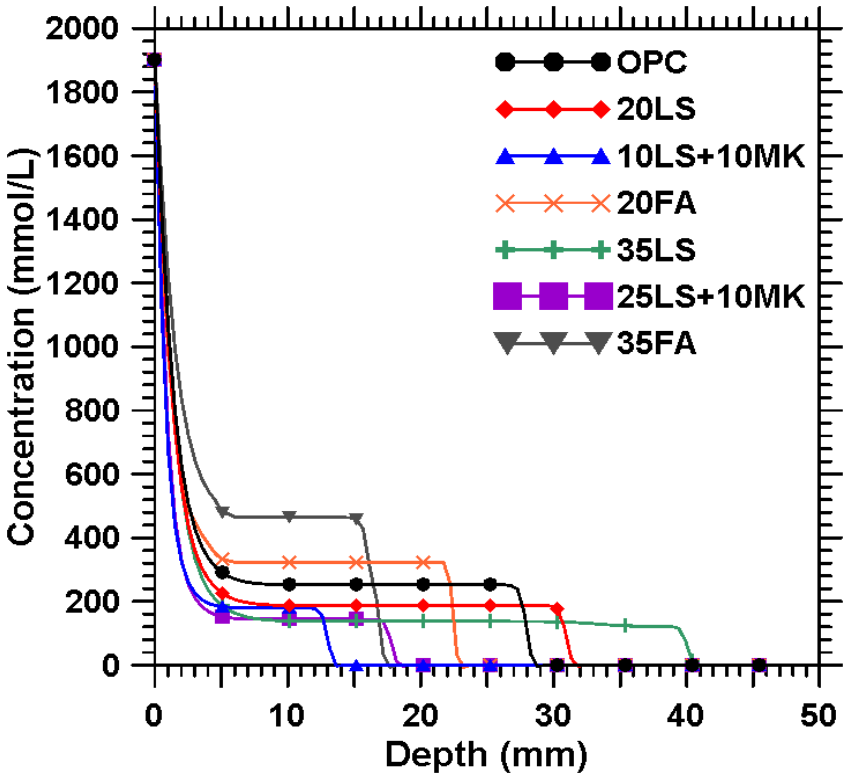


Figure 3.2. Chloride concentration profile for modified D

3.3 Concentration-Dependent Effective Diffusion Coefficients and Linear Chloride Binding

3.3.1 Chloride Binding Mechanism

Chloride binding can occur through physical adsorption of chlorides on the surface of the C-S-H gel or by chemical reaction with typically the aluminate-bearing reaction products [**22**,**23**,**24**]. Thus, total chloride ions in the system are consisted by two parts: chloride ions in liquid phase and chloride ions in solid phase. The relationship can be written as [**2**]:

$$C_t V_T = C_f V_p + C_b m_s \tag{3-7}$$

In which C_t is the total concentration of chlorides in the system, expressed as moles per cubic meter (mol/m^3); C_f is the free-chloride concentration in concrete, expressed as moles per cubic meter (mol/m^3); C_b is the bound-chloride concentration in concrete, expressed as moles per kilogram (mol/kg); V_T is the total volume of concrete, expressed as cubic meter (m^3); V_p is the volume of pore in concrete, expressed as cubic meter (m^3); m_s is the mass of solid phase, expressed as kilogram (kg).

Eq. (3-7) can be rewritten as:

$$C_t = C_f \frac{V_p}{V_T} + C_b \frac{m_s}{V_T} = C_f p + C_b (1-p)\rho_{dry} \tag{3-8}$$

Where p is the porosity of concrete and $\rho_{dry}$is the dry density of the concrete.

It has been suggested that that chloride binding can be ignored in migration tests because the rate of electrically induced ion transport is faster than the kinetics of chemical reaction or that the exposure time to chlorides in a NSSM test is too short [**16**]. However the high concentrations of chloride powder residues examined after the NSSM test in another study [**25**] indicate binding. Such binding can be described using Fruendlich

isotherm, which sketches a non-linear relationship between the bound and free chloride concentrations as Eq. (1-5), an example of which is shown in Fig. 3.3 [**4**]. However it has been shown that the Fruendlich isotherm is not appropriate for NSSM condition [**26**], and a linear binding relation is more appropriate in this case [**25**,**27**]. In this assumption, the bound chloride concentration determined by the Fruendlich isotherm is assumed to be valid at the specimen surface where is exposed to the highest Cl^- concentration, below which it linearly decreases as the mobile chloride abundance decreases (see Fig.3-3). Thus, the relationship between free and bound chloride can be described as:

$$C_b = kC_f \tag{3-9}$$

Where k is the new binding constant. k takes values ranging between 0.35×10^{-4}-to-0.5×10^{-4} [**28**] for OPC concretes and those containing OPC replacement materials, depending on the chemical composition of the replacement materials. Thus, the lower bound corresponds to binary mixtures which display suppressed binding, while the upper bound corresponds to fly ash and metakaolin bearing mixtures which display enhanced chloride binding.

To account for such Cl^- binding, Eq. (3-8) and Eq. (3-9) are substituted into Eq. (2-3) to generate a new PDE to describe the chloride transportation process:

$$\left(p + (1-p)\rho_{dry}\frac{\partial C_{Cl}^b}{\partial C_{Cl}^f}\right)\frac{\partial C_{Cl}^f}{\partial t} = \frac{\partial}{\partial x}\left(D_{Cl}\frac{\partial C_{Cl}^f}{\partial x} + D_{Cl}\frac{z_i F}{RT}C_{Cl}^f\frac{\partial \Psi(x,t)}{\partial x}\right) \tag{3-10}$$

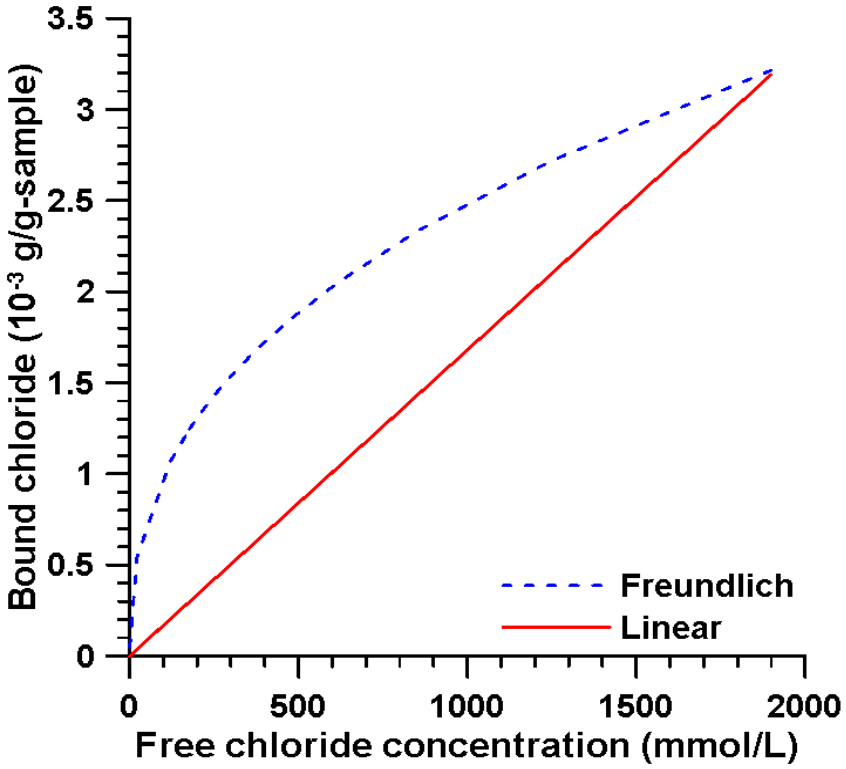


Figure 3.3. Freundlich and linear isotherms which describe Cl^- binding in cementitious mixtures

Thus, a full version of all the system equations is obtained as shown below:

$$p\frac{\partial C_{Na}}{\partial t} = \frac{\partial}{\partial x}\left(D_{Na}\frac{\partial C_{Na}}{\partial x} + D_{Na}z_{Na}\frac{F}{RT}C_{Na}\frac{\partial \Psi(x,t)}{\partial x}\right)$$

$$p\frac{\partial C_{K}}{\partial t} = \frac{\partial}{\partial x}\left(D_{K}\frac{\partial C_{K}}{\partial x} + D_{K}z_{K}\frac{F}{RT}C_{K}\frac{\partial \Psi(x,t)}{\partial x}\right)$$

$$\left(p + (1-p)\rho_{dry}\frac{\partial C_{Cl}^{b}}{\partial C_{Cl}^{f}}\right)\frac{\partial C_{Cl}^{f}}{\partial t} = \frac{\partial}{\partial x}\left(D_{Cl}\frac{\partial C_{Cl}^{f}}{\partial x} + D_{Cl}\frac{z_i F}{RT}C_{Cl}^{f}\frac{\partial \Psi(x,t)}{\partial x}\right) \quad (3\text{-}11)$$

$$p\frac{\partial C_{OH}}{\partial t} = \frac{\partial}{\partial x}\left(D_{OH}\frac{\partial C_{OH}}{\partial x} + D_{OH}z_{OH}\frac{F}{RT}C_{OH}\frac{\partial \Psi(x,t)}{\partial x}\right)$$

$$\nabla^2\Psi = -\frac{F}{\varepsilon_0\varepsilon_r}\sum_i C_i z_i$$

3.3.2 Concentration-Dependent Effective Diffusion Coefficients

To consider the concentration dependence of the ionic diffusion coefficients, an approach adopted by [**5**] is used, as noted in Eq. (3-12):

$$\begin{cases} D = D_{0.01M} = p\beta D^{inf} & (\mathrm{c} < 0.01\ \mathrm{mol/L}) \\ D = D_{1M} c^{a} & (0.01\mathrm{mol/L} < \mathrm{c} < 1\mathrm{mol/L}) \\ D = D_{1M} & (\mathrm{c} > 1\mathrm{mol/L}) \end{cases} \tag{3-12}$$

In which *a* is an exponent equal to -0.71 for the mixtures used in this study.

Based on these modifications, numerical simulations for all specimens were repeated.

3.4 Effect of Chemical Activity

As mentioned in Section 2.1, it is assumed that the influence from chemical activity is negligible during the electro-migration test. In this section, detailed procedure is provided to validate the assumption.

3.4.1 Modeling the Chemical Activity

In many past works, several models have been developed to calculate the chemical activity coefficient based on the ionic concentration. However, models such as Debye-Hückel or extended Debye- Hückel or Davies are unable to describe the thermodynamic behavior of highly concentrated ionic solution properly [**16**, **29**], a modified Davies equation is applied to calculate the chemical activity coefficient [9**]:**

$$\ln \gamma_i = -\frac{Az_i^2\sqrt{I}}{1 + a_i B\sqrt{I}} + \frac{(0.2 - (4.17 \times 10^{-15})I)Az_i^2 I}{\sqrt{1000}} \tag{3-13}$$

In which *I* is the ionic strength of the solution, which can be calculated from Eq. (3-5).

A and B are temperature dependent parameters that can be obtained from the following equations:

$$A = \frac{\sqrt{2}F^2 e_0}{8\pi(\varepsilon_r \varepsilon_0 RT)^{3/2}} \tag{3-14}$$

$$B = \sqrt{\frac{2F^2}{\varepsilon_r \varepsilon_0 RT}} \tag{3-15}$$

Where e_0 is the electronic charge (1.602×10^{-19} C), $\varepsilon_r\varepsilon_0$ is the permittivity of the medium, which can be found in Section 2.1, F, R and T also have the same meaning as defined in Section 2.1. a_i represents the radius of ion *i*, as given in Table 3-5.

Table 3.5 Radius of different ions (10^{-12} m)

Na^+	K^+	OH^-	Cl^-
116	152	110	167

Based on Eq. (2-15), induce the fourth matrix $\boldsymbol{A_5}$ to include the effect from chemical activity into the system equation. $\boldsymbol{A_5}$ is given as:

$$\boldsymbol{A_5} = \begin{bmatrix} D_1 \dfrac{\partial \ln \gamma_1}{\partial x} & D_2 \dfrac{\partial \ln \gamma_2}{\partial x} & D_3 \dfrac{\partial \ln \gamma_3}{\partial x} & D_4 \dfrac{\partial \ln \gamma_4}{\partial x} & 0 \end{bmatrix} \tag{3-16}$$

The updated system stiffness matrix is given as:

$$\boldsymbol{K} = \int (\{\boldsymbol{n}\}^T\{\boldsymbol{A_2}\}\{\boldsymbol{n}\} + \{\boldsymbol{N}\}^T\{\boldsymbol{A_3}\}\{\boldsymbol{N}\} + \{\boldsymbol{n}\}^T\{\boldsymbol{A_4}\}\{\boldsymbol{n}\} + \{\boldsymbol{n}\}^T\{\boldsymbol{A_5}\}\{\boldsymbol{N}\})\, dx \tag{3-17}$$

Following the same procedure introduced in Section 2.2, the chemical activity included simulation can be carried out.

3.4.2 Simulation Results and Discussion

To validate the feeble effect of chemical activity on the electro-diffusion process, one simulation is carried out. Fig. 3.4 shows the chloride fluxes contributed by different mechanisms after 24-hours simulation. It is obvious that during the NSSM test, migration dominates the penetration process. Compared with migration, contribution from chemical

activity is much smaller (2-3 magnitudes smaller than migration), which indicates that the assumption is actually acceptable. Samson et al also got the same conclusion by comparing the effect of chemical activity in different solutions [**9**].

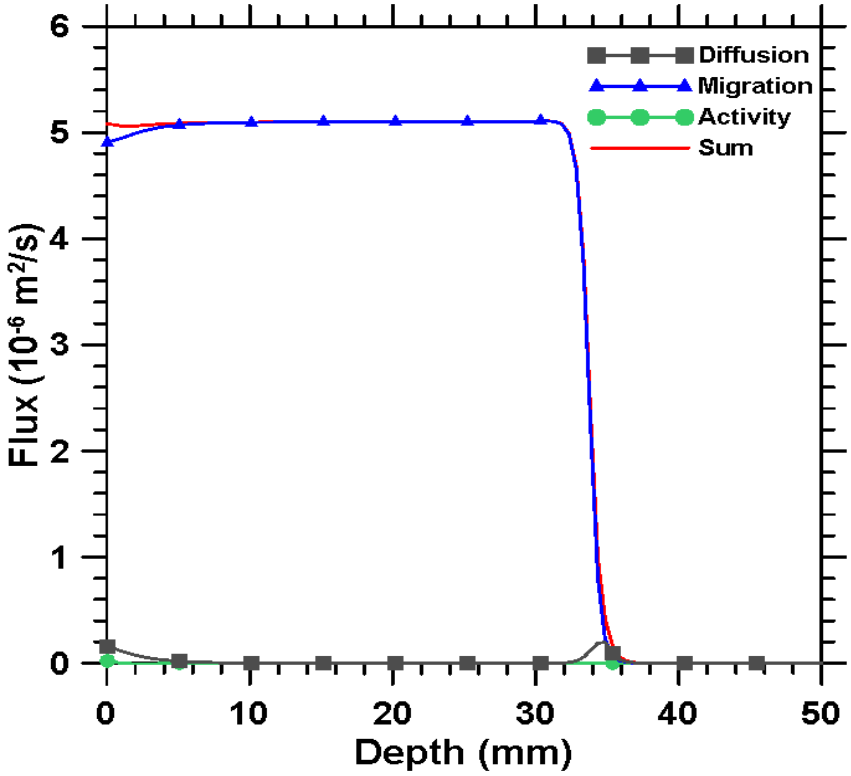


Figure 3.4. Chloride fluxes profile

4 NUMERICAL SIMULATION RESULTS AND DISCUSSION

Based on Eq. (3-11) and Eq. (3-12), simulations of the NSSM test were implemented for all the concrete specimens which were cured for 56 days. The initial and boundary conditions are listed in Table 4.1.

Table 4.1 Pore structure factors and the initial and boundary conditions for the simulations

Mixture	φ	β	Pore solution composition (mol/L)			Upstream boundary conditions (mol/L)		Downstream boundary conditions (mol/L)	
			K+	Na+	OH-	Na+	Cl-	Na+	OH-
OPC	0.099	0.028	0.21	0.14	0.35				
LS20	0.122	0.030	0.16	0.11	0.27				
LS10+MK10	0.104	0.010	0.16	0.11	0.27				
LS35	0.132	0.040	0.12	0.08	0.20	1.9	1.9	0.3	0.3
LS25+MK10	0.129	0.014	0.13	0.09	0.22				
FA20	0.127	0.014	0.25	0.30	0.55				
FA35	0.132	0.009	0.27	0.40	0.67				

Fig. 4.1 gives the chloride concentration profile after 24 hours of the NSSM test for all the seven cases given in Table 4.1. The dotted line indicates the threshold value of chloride (70 mmol/L) at which the free chloride ions can be measured by the colorimetric method (spraying silver nitrate solution mentioned earlier). It is obvious from this figure and Table 4.1 that the pore structure parameter, which includes porosity and pore connectivity, dominates the speed of chloride penetration process into concrete. The mixture with 35% of cement replaced by limestone, having both the highest porosity and pore connectivity has a largest penetration depth while the mixture with 35% of cement replaced by fly ash has the lowest product value of porosity and pore connectivity, and correspondingly the smallest calculated penetration depth. It is also noticed that the initial pore solution composition determines the maximum stable chloride level that can be

reached during the NSSM test. That is due to the restriction imposed by the electroneutrality condition (Poisson's equation). Chloride ions together with the other ions in solution, mainly sodium and potassium, equilibrate to maintain the electroneutrality condition.

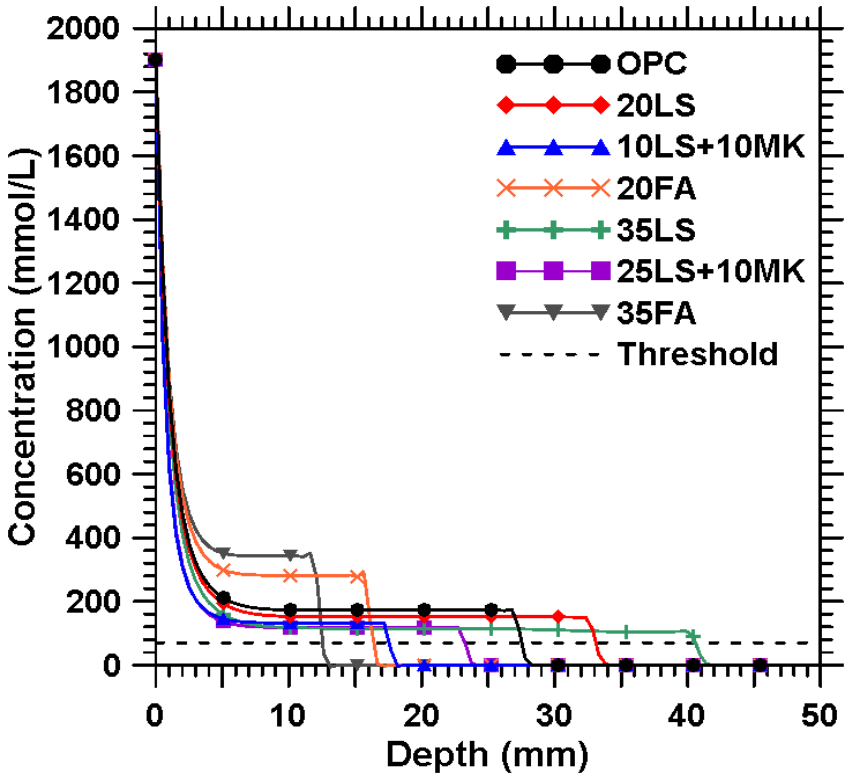


Figure 4.1. Chloride concentration profile @ 24 hours

Fig. 4.2 describes the relationship between the simulated penetration depth of chloride ions and the experimentally measured results. Favorable correlations are observed. This is significant in that, suitable numerical implementations of the PNP solutions can be applied to parametrically evaluate the behaviors of various concrete formulations to discern and rank the effects of different parameters including: (a) porosity, (b) pore structure factor, (c) solid binder and the pore solution composition, (d) external Cl^- concentration in solution, (e) applied voltage, and (f) test duration on ionic transport resistance.

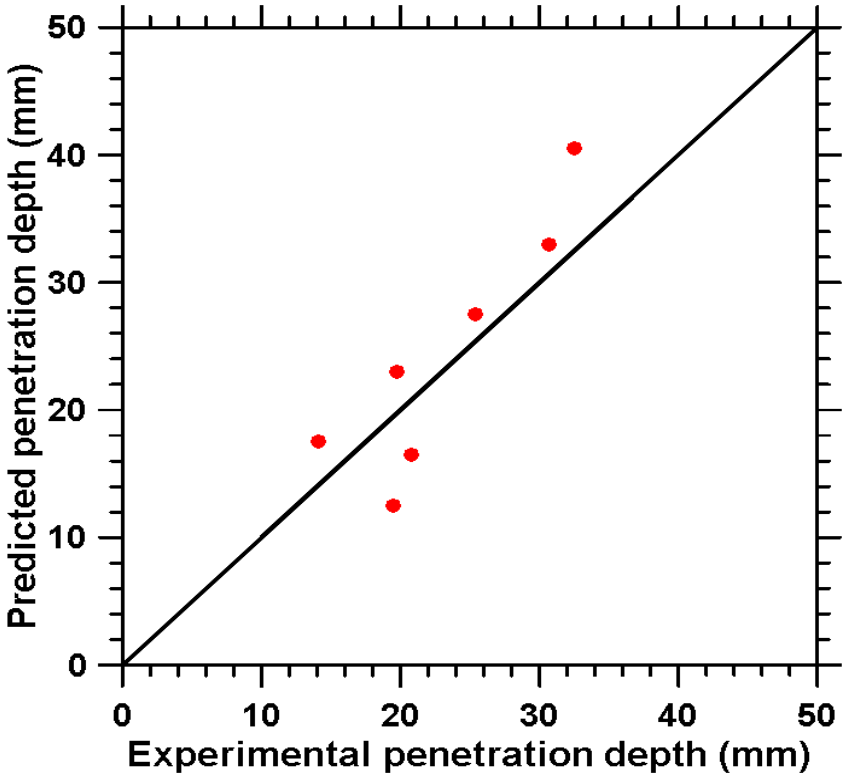


Figure 4.2. The relationship between the experimentally measured Cl^- penetration depths and those simulated from the modified PNP formulations for binary and ternary concrete mixtures

Fig. 4.3 provides concentration profile of different ions at different times during the NSSM test. While similar curves can be generated for all the concrete mixtures studied here, only the data for OPC concrete is demonstrated in order to explain the trends in a concise fashion. These four curves give a general idea that during the test, under the effect of external potential, the cations (potassium and sodium) move upstream while the anions (chloride and hydroxyl) move downstream. It should be noted that the simulations predict a sharp drop in Cl^- profiles very close to the exposed surface, after which a stable regime follows, and then the Cl^- ion abundance drops once again. This is likely on account of the exaggerated Cl^- loading in the exposure solution, and the enforcement of electroneutrality in the simulations, which leads to a mismatch in the early Cl^- penetration depths as compared to experimental determinations of Cl^- intrusion using

gravimetric techniques. Several reasons can be ascribed to this observation. Firstly, there may exist local violations of the electroneutrality condition, especially very close to the specimen surface, where diffusion and Cl^- binding may occur simultaneously. Secondly, the large amount of Cl^- ingress close to the exposed face results in an increased level of binding predicted by the model, while in reality, the amount of bound Cl^- may be much smaller.

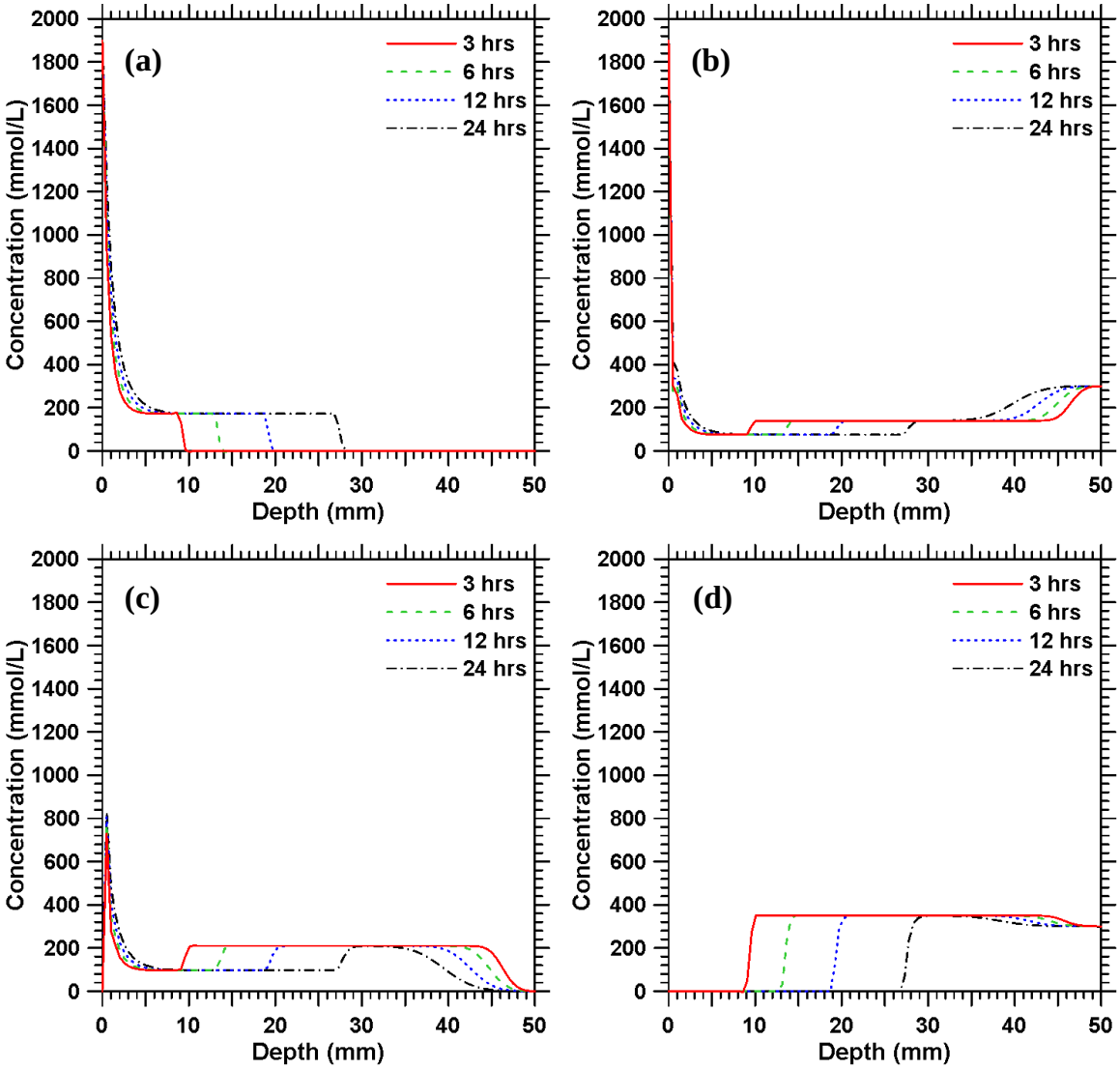


Figure 4.3. Distribution of concentrations of ions of OPC at different times: (a) Chloride; (b) Sodium; (c) Potassium; (d) Hydroxyl

Fig. 4.4 gives the electrical potential profile at different times for the OPC concrete. From this figure, it is obvious that during the NSSM test, due to the influence of the membrane potential generated by different ionic transport speeds, the potential drop is no more in a linear relationship. On the bi-linear response relating electrical potential to specimen depth shown in this figure, the point at which the response changes its slope indicates the depth of chloride penetration at that particular time. Before this point, due to the introduced chloride ions, concentration levels of both the cations and anions change significantly and a strong membrane potential is generated that influence the external potential, which may also accelerate the penetration process. After this intersection point, since there are no external ions to redistribute the existing ionic balance system, the membrane potential can be negligible.

This phenomenon can also be explained using Fig. 4.5, which is the electrical field profile of the OPC concrete undergoing NSSM test at different times. The trend in this figure can be described as an "initial increase – stable – sharp drop" process. The sharp drop front of the curve corresponds to the intersection point in Fig. 4.4 and also denotes the chloride penetration depth. It also should be noted that at an early age during the test, i.e. 3 hours, on account of the drastic change in ionic concentrations, a larger electrical field is generated, which also gives a higher slope to the potential – depth curve. As the test goes on, this effect is reduced because the ionic concentration changes are gradual.

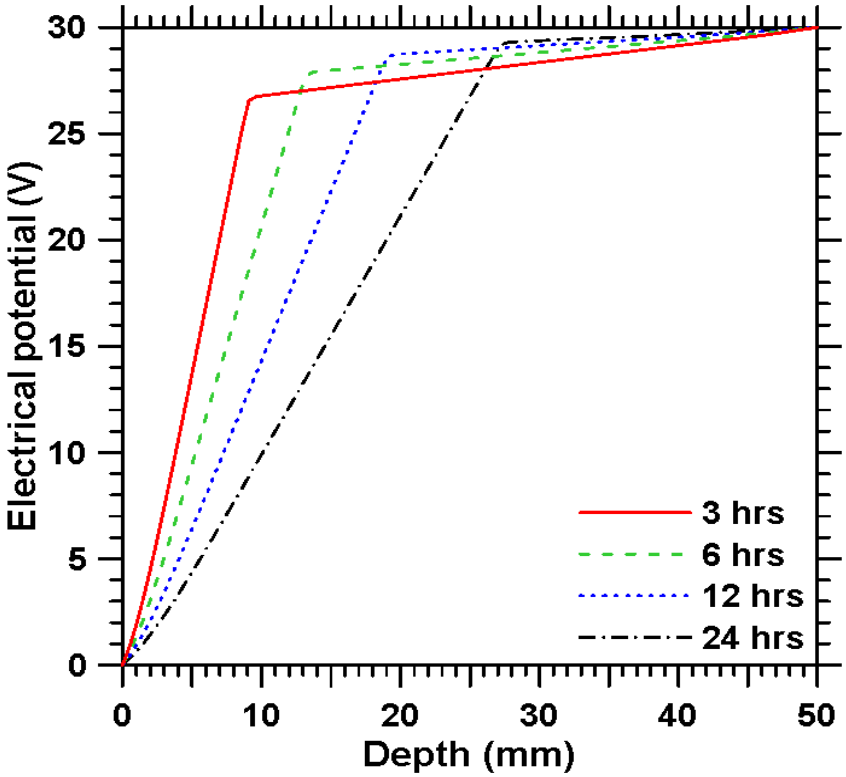


Figure 4.4. Electrical potential profile of OPC at different times

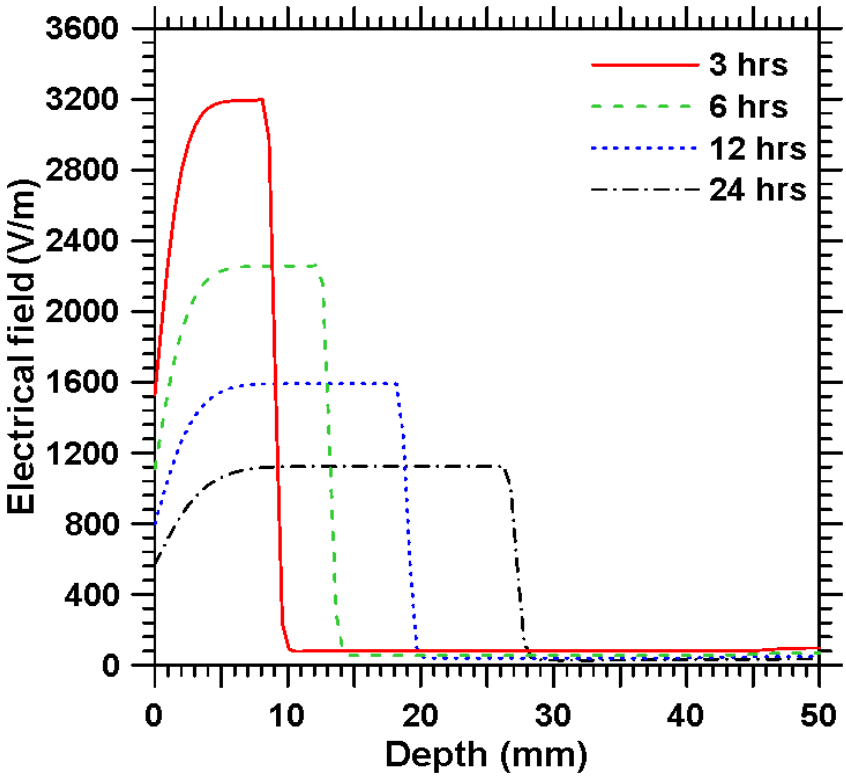


Figure 4.5. Electrical field profile of OPC at different times

Fig. 4.6 provides the relationships between concentration and flux of different ions for OPC concretes after 24 hours of NSSM test. It can be noticed from Fig. 4.3 that chloride and hydroxyl ions have a synchronous movement, when chloride penetrates into certain depth, hydroxyl totally moves out from the penetrated part. This is partly the result of the need to maintain electroneutrality at all times, at all locations. As a result, linear relationships between ionic flux and concentration are found for anions while separated two stages of linear relationships are observed for cations. During the electro-migration process, as chloride penetrating into the media to a fixed depth, hydroxyl concentration drops to zero up to this level, cations have to redistribute themselves to balance with chloride ions. Beyond this level, the ionic concentrations change with a smaller rate – sodium and hydroxyl entre the media from the downstream solution with a much lower concentration compared with upstream solution while potassium continuously leaching out from the media. As it is showed in Fig. 4.6(a), during the chloride penetrating process, with the drop of concentration level, flux value also drops until zero – as the chloride stops at the sharp front as shown in Fig. 4.3(a). In Fig. 4.6(b) and Fig. 4.6(c), as the ionic concentrations drop, there're corresponding drop for ionic fluxes, this describes the status for chloride-penetrated part. After a short constent concentration status, which indicates the stable level as showed in Fig. 4.3, there're again linear relationships for sodium and potassium with a much smaller slopes. These two curves together with Fig. 4.6(d), gives the status beyond the chloride-penetrated part, in which ions have smaller fluxes. Fig. 4.6 also gives the general idea that during the NSSM test, migration dominates the process with a strong external electrical potential, which can be found in previous section.

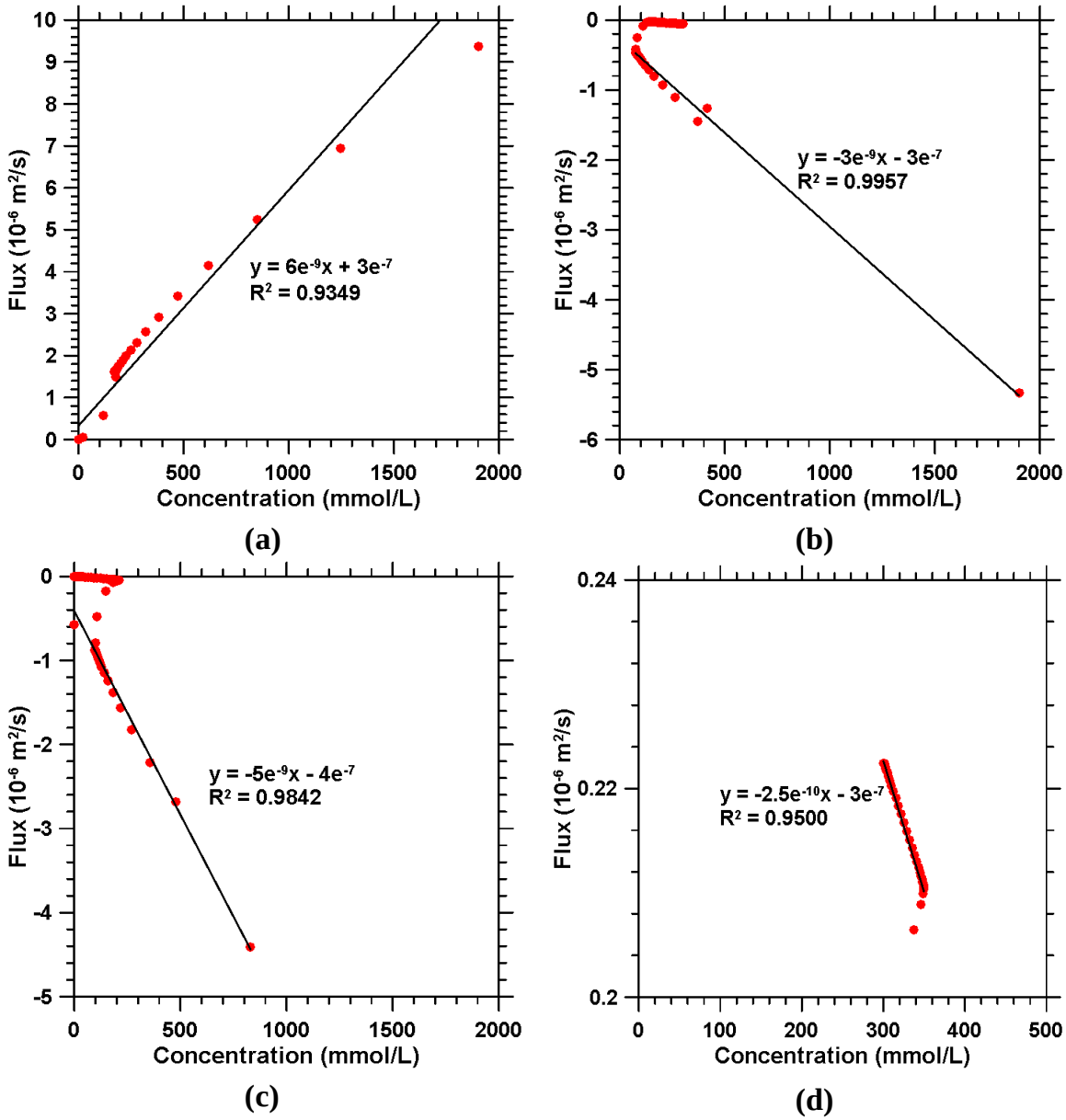


Figure 4.6. Flux-concentration relationship of different ions for OPC @ 24 hours NSSM simulation: (a) Chloride; (b) Sodium; (c) Potassium; (d) Hydroxyl

Fig. 4.7 describes the chloride penetration process with testing times for OPC and 20% OPC replacement concrete. It can be found that during the NSSM test, the chloride penetrating speed varies with time. The penetration process starts with a higher speed and slows down until a stable level. One reason for that phenomenon is the chloride binding mechanism. Initially, binding does not happen in the solution since the chloride content is

little. Once the free chloride content reaches a threshold level, e.g. 0.14% by mass of concrete [**26**], chloride ions start bind to solid phases, which retards the penetration process. Compared the four cases showed in the figure, 20% limestone replacement concrete has a largest penetration depth with a largest penetrating speed since it has larger porosity, pore connectivity together with a smallest binding ability. While 10% limestone 10% metakaolin replacement concrete and 20% fly ash replacement concrete have stronger ability to bind more chloride ions, these two cases have more apparent drop in penetrating speed.

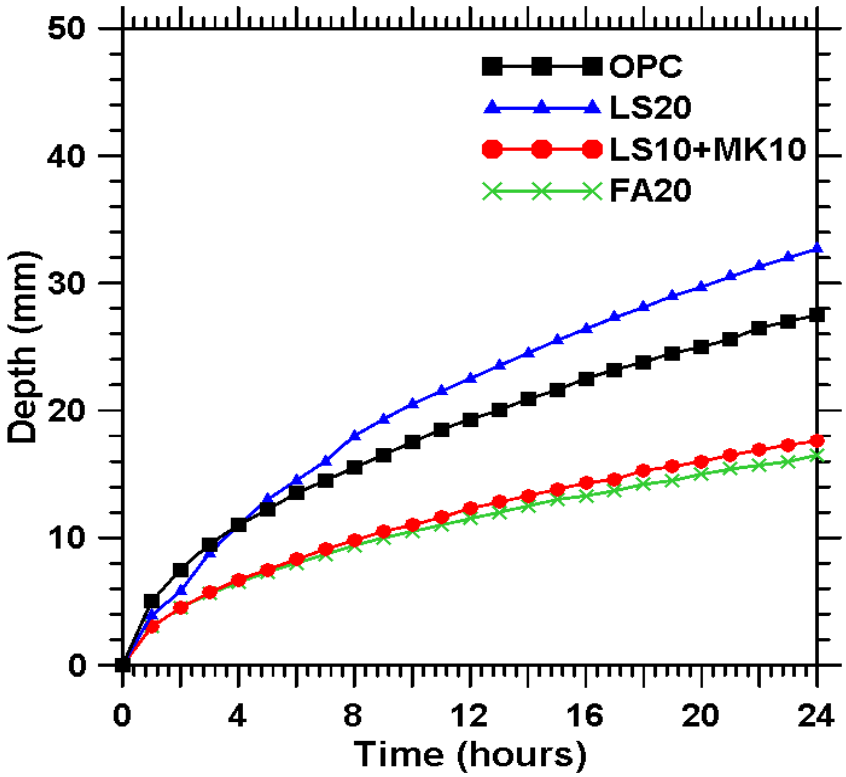


Figure 4.7. Chloride penetration development for OPC

5 SUMMARY

Numerical simulation is a powerful method to implement a virtual analysis of ionic transport problems. By simulating non-steady state migration test, the process of ionic transport under electrically induced conditions is explored for OPC and modified concrete systems. The governing equation of the system is a convection-diffusion equation, and the accuracy of its solution is heavily dependent on both the element number and time step used in simulation. In this study, a total of 1000 elements and a time step of one second were chosen as default values for the models. While decreasing the numbers of elements leads to oscillatory results, increasing the value of time step results in totally unstable results. By using the Petrov-Garlerkin method, unconditional stable solution can be reached without significantly losing accuracy. All the numerical simulation codes are included in Appendix A.

When there's strong external electrical voltage exists in the system, diffusion part can be omitted during simulation since the electrical migration dominates the penetration process. However, the specific relationship between diffusion and electrical migration is still unclear. Further work is needed to explore how electrical field influence the ionic diffusion process. During the NSSM test, the ionic diffusion coefficients varies with the ionic concentration, a higher concentration level gives a slower diffusion speed. Porosity, pore connectivity and the binder's chloride binding ability influence the penetration process. Larger porosity and pore connectivity, smaller binding ability lead to higher penetration depth. It is also noticed that the assumption of constant electrical field is improper. During the NSSM, electrical filed varies with the changing of ionic concentration. It should be point out that the predicted concentration profile differs from

the experimental profile at the penetrated part, which is also need more further work to explore the reason. It is also noticed that the chloride penetration spped varies with testing time. As time goes on, increasing of chloride ion concentration level and increasing of binding slow the penetration process.

PART II. COMPUTATIONAL EVALUATION OF THE INFLUENCE OF CRACK PROPERTIES ON CONCRETE PERMEABILITY

6 INTRODUCTION

The permeability of concrete has important implications on its durability since permeability controls the rate of penetration of moisture that may contain aggressive solutes and also controls moisture movement during heating and cooling or freezing and thawing [**30**]. During the past several decades, different methods were developed for a better understanding of concrete permeability. These models generally consider uncracked concretes. Extensive research has been done [**31**,**32**,**33**,**34**,**35**] on the water permeability of crack-free concrete and this leads to the general conclusion that the saturated water permeability of concrete is a function of its porosity, pore connectivity, and the square of a threshold pore diameter **[33**,**34**,**35]**, which is similar to what Katz and Thompson developed and is a well-accepted permeability model for rocks. On the other hand, research on transport through cracked concrete has been limited. The major reason for this is that characterizing cracking in concrete is rather difficult and identifying the parameters of the crack that influence permeability is non-trivial. The pioneering works of Kermani [**36**], Tsukamoto and Wörner [**37**], and Gérard et al. [**38**] explored changes in permeability of concrete caused by the application of compressive or tensile stress. Akhavan et al. [**39**] explored the effect of geometric parameters of cracks on permeability. In this part of this thesis, the focus is on numerically modeling the moisture transport through concretes containing cracks of varying sizes, shapes, and tortuosity. A modified Louis equation is chosen as the primary model to estimate the crack permeability from its geometric properties. As a comparison, Navier-Stokes equation and Lattice-Boltzmann

method are also studied and discussion about the differences is indicated. This study is a preliminary effect to discern the geometric effects of cracks on permeability through numerical simulations.

7 MATHEMATICAL MODELS FOR PERMEABILITY ESTIMATION

7.1 Modified Louis Equation

Modified Louis equation is an expression that connects the permeability with geometric properties of crack – effective crack width, tortuosity and surface roughness [**39**]:

$$\kappa = \frac{\tau w^2}{12(1 + 8.8R_r^{1.5})} \tag{7-1}$$

In which $\kappa\ (m^2)$ is the crack permeability, $w\ (m)$ is the effective crack width, τ is the tortuosity factor and R_r is the relative surface roughness of a crack.

7.1.1 Effective Crack Width

In this study, the developed model assumes that the crack propagates into several layers along depth of concrete with unit thickness in perpendicular direction. At each layer, crack path consists of circles with different radii along a zig-zag line. Thus, the basic crack is generated as shown in Fig. 7.1.

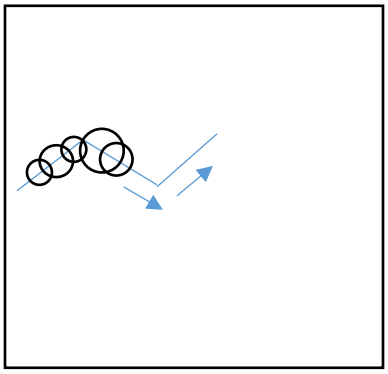


Figure 7.1. Crack generation model (Top surface)

According to Darcy's law, for the first row (surface), the volumetric discharge rate ($Q_{1,T}$) is described as:

$$Q_{1,T} = \sum_{i=1}^{n} Q_{1,i} = \frac{1}{\eta} \sum_{i=1}^{n} L_{1i} w_{1i} k_{1i} \frac{\Delta P_{1i}}{d_{1i}} \tag{7-2}$$

In which n is the total number of circles in each layer, w_{1i} is the representative width of each circle, $k_{1i} = \frac{{w_{1i}}^2}{12}$ and ΔP_{1i} represent the permeability [**40**] and pressure loss for each element. Assuming that the elements' length and thickness are chosen to be constants: $L_{ij} = L$ and $d_{ij} = d$, and that the flow is one-dimensional ($\Delta P_{11} = \Delta P_{12} = \cdots = \Delta P_1$):

$$Q_{1,T} = \frac{1}{12\eta}\frac{L}{d}\Delta P_1 \sum_{i=1}^{n} {w_{1i}}^3 \tag{7-3}$$

Combining Eq. (2-3) with Darcy's law results in:

$$Q_{1,T} = \frac{1}{12\eta}\frac{nL}{d}\Delta P_1 {w_{1-eff}}^3 \tag{7-4}$$

$$w_{1-eff} = \sqrt[3]{\frac{1}{n}\sum_{i=1}^{n} {w_{1i}}^3} \tag{7-5}$$

Finally, the effective crack width of each layer can be expressed as:

$$w_{j-eff} = \sqrt[3]{\frac{1}{n}\sum_{i=1}^{n} {w_{ji}}^3} \tag{7-6}$$

In which w_{ji} is the representative width of *i*-th circle in *j*-th layer, w_{j-eff} is the effective crack width of *j*-th layer.

To calculate the total effective through-crack width, w_{eff}, assume $Q_T = Q_{1,T} = Q_{2,T} = \cdots = Q_{j,T}$,

Then based on Eq. (2-4),

$$\Delta P_1 {w_{1-eff}}^3 = \Delta P_2 {w_{2-eff}}^3 = \cdots = \Delta P_j {w_{j-eff}}^3 \tag{7-7}$$

Where Q_T is the total discharge rate, $\Delta P = \Delta P_1 + \cdots + \Delta P_j$ is the total pressure loss across the specimen, and *j* is the number of layers.

Then

$$\Delta P_i = \Delta P_1 \left(\frac{w_{1-eff}}{w_{i-eff}}\right)^3 \tag{7-8}$$

$$\Delta P = \sum_{i=1}^{j} \Delta P_i = \Delta P_1 {w_{1-eff}}^3 \sum_{i=1}^{j} \left(\frac{1}{{w_{i-eff}}^3}\right) = j\Delta P_1 {w_{1-eff}}^3 \left(\frac{1}{{w_{eff}}^3}\right) \tag{7-9}$$

And ultimately,

$$w_{eff} = \sqrt[3]{\frac{j}{\sum_{i=1}^{j}\left(\frac{1}{{w_{i-eff}}^3}\right)}} \tag{7-10}$$

7.1.2 Crack Tortuosity

The tortuosity of crack can be easily determined as:

$$\tau = \left(\frac{X}{L_e}\right)^2 \tag{7-11}$$

In the above equation, X is the depth of specimen and L_e is the actual crack length. It has been shown in [**41**] that permeability drops proportionally with $\left(\frac{X}{L_e}\right)^2$ but not with $\frac{X}{L_e}$ since the larger effective length affects both pressure gradient and fluid velocity.

7.1.3 Crack Roughness

As shown in Fig. 7.2, roughness is determined in two steps in this study. First, the segment $x = 0$ to $x = \lambda$ is selected and its roughness is determined by calculating the average height of surface asperities with respect to its reference $\bar{Z}$ line as:

$$R_{a.l} = \frac{1}{\lambda}\sum(|\bar{Z}(x) - Z(x)|\cos\alpha) \tag{7-12}$$

In which $R_{a.l}$ is the local roughness over this segment, and the quantity in front of Σ is the absolute value of the difference between crack profile and the reference line in the direction perpendicular to the reference line. Next, the segment is shifted 1 pixel to the right ($x = 1$ to $+1$) and the local roughness is recalculated. The segment is swept over the entire assessment length ($x = 0$ to $x = X_{max}$) and the corresponding $R_{a.l}$ values are calculated. A total of ($X_{max} - \lambda$) number of $R_{a.l}$ values are averaged to determine the global surface roughness:

$$R_{a.g} = \frac{1}{X_{max} - \lambda} \sum_{i=1}^{X_{max}-\lambda} (R_{a.l})_i \tag{7-13}$$

Here, $R_r = R_{a.g}/2w_{eff}$ is the relative surface roughness.

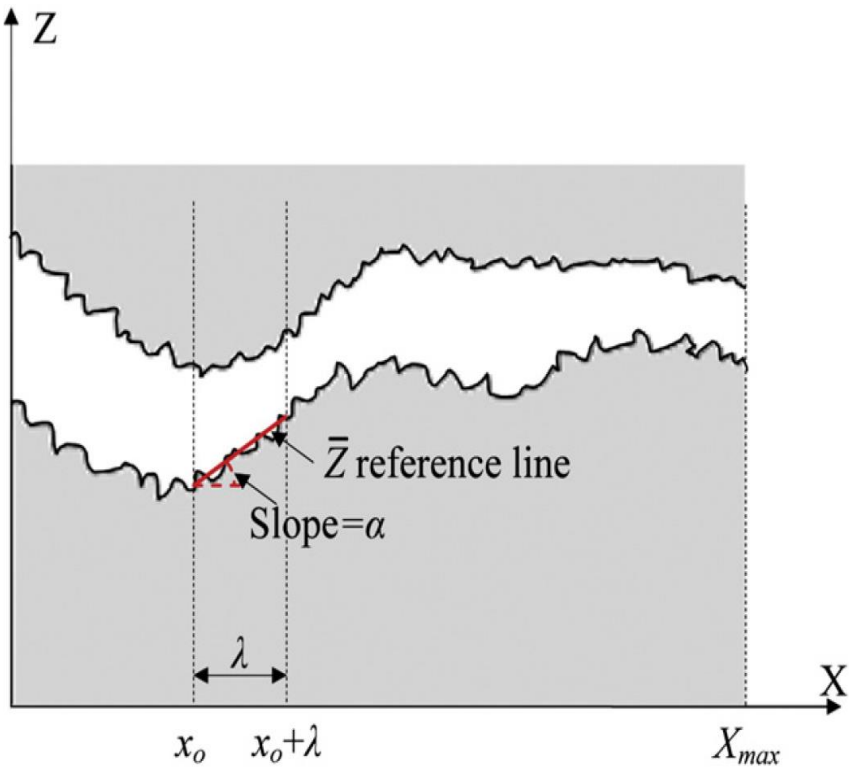


Figure 7.2. Schematic of a crack profile to illustrate the method of quantifying tortuosity and roughness [**39**]

7.2 Navier-Stokes Equations

Absolute permeability appears in Darcy's law as a constant coefficient relating fluid, flow and material parameters:

$$\frac{Q}{A} = \frac{k}{\mu}\frac{\Delta P}{dZ} \tag{7-14}$$

In the above equation, Q (m^3/s) is the global flow rate through the crack, A (m^2) is the cross section of the crack, k (m^2) is the absolute permeability, μ $(Pa \cdot s)$ is the dynamic viscosity of the flowing fluid and $\left[\frac{\Delta P}{dZ}\right]$ (Pa/m) is the pressure gradient.

To numerically estimate the absolute permeability, the simplified Navier-Stokes equations as given below can be solved:

$$\begin{cases} \nabla \boldsymbol{u} = 0 \\ \mu \nabla^2 \boldsymbol{u} - \nabla P = \mathbf{0} \end{cases} \tag{7-15}$$

In which $\boldsymbol{u}$ is the velocity of the fluid; P is the pressure of the fluid and μ is the dynamic viscosity of the flowing fluid.

The simplification is based on the following considerations: 1) Incompressible fluid, which means constant density of the fluid; 2) Newtonian nature of the fluid, which gives a constant dynamic viscosity; 3) Steady-state flow, which indicates that the velocity does not vary over time; and 4) Laminar flow, which means that the concerned velocities are small enough not to produce turbulence [**42**].

7.3 Lattice Boltzmann Methods

Lattice Boltzmann methods are special numerical schemes for solving the incompressible Navier-Stokes equations. The set of equations are given as below:

$$\nabla \boldsymbol{u} = 0$$
$$\partial_t(\rho \boldsymbol{u}) + \nabla(\rho \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla p + \mu \nabla^2 \boldsymbol{u} \tag{7-16}$$

In which $\boldsymbol{u}$ stands for the fluid velocity, ρ represents density, p denotes pressure, μ represents viscosity, and $\otimes$ stands for the tensor product of two vectors. Fluids fulfilling those equations for a constant viscosity are called Newtonian fluids, all others are referred to as non-Newtonian.

In Lattice-Boltzmann methods, instead of discretizing the Navier-Stokes equation directly, particle dynamics is simulated on a mesoscopic scale. Compared with traditional methods, they are a serious alternative option for computational fluid dynamics based modeling [**43**,**44**,**45**,**46**]. At each time *t*, consider the concentration of particles $f_i(\boldsymbol{x}, t)$ located at lattice node $\boldsymbol{x}$ and moving with lattice velocity $\boldsymbol{v}$, where $\boldsymbol{v}$ can only take certain constant values that make sure the particle density is moving from one lattice point to another during one time step Δt. The general Lattice Boltzmann equation is expressed as:

$$f_i(\boldsymbol{x} + \boldsymbol{v_i}\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + \Omega_i \tag{7-17}$$

where Ω_i represents the so-called collision operator. Several definitions have been given for the collision operator, each of them defining a special Lattice Boltzmann scheme.

One of the widely used approximations is the Bhatnagar-Gross-Krook (BGK) model [**47**]. They noticed that the main effect of the collision operator is to bring the velocity distribution function closer to the equilibrium distribution. Based on the BGK approximation [**47**,**48**], the collision operator can be defined as:

$$\Omega_i = -\frac{1}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\right) \tag{7-18}$$

In which τ is the rate of relaxation towards local equilibrium, $f_i^{eq}(\boldsymbol{x}, t)$ is the equilibrium distribution function.

In this thesis, two-dimensional square lattice with nine-velocity (D2Q9) BGK model is applied since it is successful for simulations of two-dimensional flows [**49**]. The schematics can be seen in Fig. 7.3.

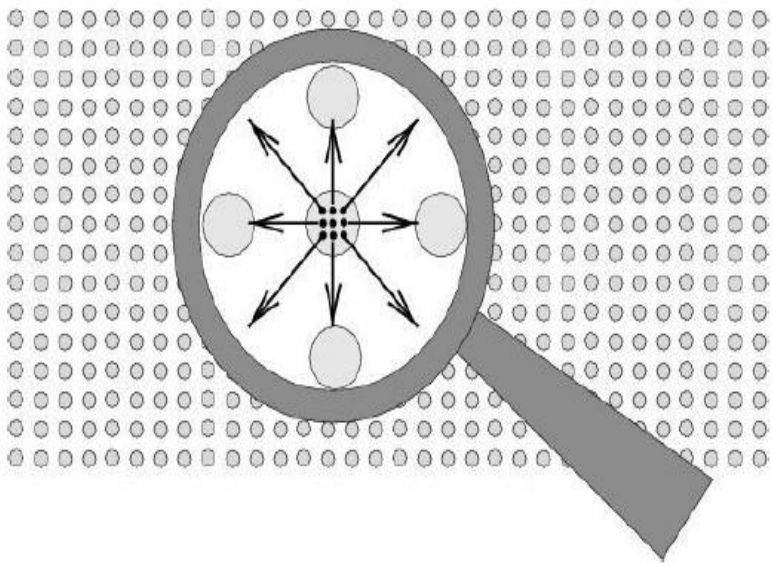


Figure 7.3. D2Q9 model description for Lattice Boltzmann simulations

The velocity is defined as:

$$\boldsymbol{v_i} = c\boldsymbol{e_i} \tag{7-19}$$

Where $c = \Delta x/\Delta t$, Δx and Δt are the lattice grid spacing and time step. $\boldsymbol{e_i}$ is the velocity directions given as:

$$\boldsymbol{e_i} = \begin{cases} (0,0) & i = 0 \\ (\cos[(i-1)\pi/2], \sin[(i-1)\pi/2]) & i = 1,2,3,4 \\ \sqrt{2}(\cos[(i-5)\pi/2+\pi/4], \sin[(i-5)\pi/2+\pi/4]) & i = 5,6,7,8 \end{cases} \tag{7-20}$$

The relaxation parameter τ determines the kinematic viscosity ν of the simulated fluid:

$$\nu = (\tau - 0.5)C_s^2\Delta t \tag{7-21}$$

Where $C_s = c/\sqrt{3}$ is the speed of sound.

The equilibrium distribution function $f_i^{eq}(\boldsymbol{x}, t)$ is defined as:

$$f_i^{eq}(\boldsymbol{x},t)=\begin{cases}-4\sigma\dfrac{p}{c^2}+s_0\big(\boldsymbol{u}(\boldsymbol{x},t)\big) & i=0\\ \lambda\dfrac{p}{c^2}+s_i\big(\boldsymbol{u}(\boldsymbol{x},t)\big) & i=1,2,3,4\\ \gamma\dfrac{p}{c^2}+s_i\big(\boldsymbol{u}(\boldsymbol{x},t)\big) & i=5,6,7,8\end{cases} \tag{7-22}$$

Where

$$s_i\big(\boldsymbol{u}(\boldsymbol{x},t)\big)=w_i\left[3\frac{\big(\boldsymbol{e_i}\cdot\boldsymbol{u}(\boldsymbol{x},t)\big)}{c}+4.5\frac{\big(\boldsymbol{e_i}\cdot\boldsymbol{u}(\boldsymbol{x},t)\big)^2}{c^2}-1.5\frac{|\boldsymbol{u}(\boldsymbol{x},t)|^2}{c^2}\right] \tag{7-23}$$

with the weight coefficient

$$w_{\boldsymbol{i}}=\begin{cases}\dfrac{4}{9} & i=0\\ \dfrac{1}{9} & i=1,2,3,4\\ \dfrac{1}{36} & i=5,6,7,8\end{cases} \tag{7-24}$$

σ, λ and γ are parameters satisfying $\lambda+\gamma=\sigma$, $\lambda+2\gamma=1$.

The distribution function satisfies the following conservation laws:

$$\sum f_i(\boldsymbol{x},t)=\sum f_i^{eq}(\boldsymbol{x},t) \tag{7-25}$$

$$\sum \boldsymbol{v_i} f_i(\boldsymbol{x},t)=\sum \boldsymbol{v_i} f_i^{eq}(\boldsymbol{x},t) \tag{7-26}$$

The local fluid pressure and velocity are given by:

$$p=\frac{c^2}{4\sigma}\Big[\sum f_i(\boldsymbol{x},t)+s_0\big(\boldsymbol{u}(\boldsymbol{x},t)\big)\Big] \tag{7-27}$$

$$\boldsymbol{u}(\boldsymbol{x},t)=\sum \boldsymbol{v_i} f_i(\boldsymbol{x},t) \tag{7-28}$$

8 GENERATION OF SINGLE 3D CRACK

For the permeability analysis methods described in this thesis, a controllable random 3D crack is necessary as a starting point. As discussed in an earlier section, 3D crack was constructed using 2D square images having a size of 300 pixels x 300 pixels and a total depth of 300 pixels.

8.1 Surface 2D Crack

Firstly, the surface crack shape was built as a basic model for other layers. Cracks on different layers are assumed to have the same shape as the surface crack, but they are of different crack widths and positions in the layer. Another assumption is that the basic crack has a zig-zag shape. First of all, the total numbers of zig-zags are defined and the starting point of the crack chosen. A certain number of random numbers are generated and they are assigned as slopes for those zig-zag lines. A scale factor was used to adjust the slopes if necessary. Another set of random numbers were generated (between 0.5 and 1) for crack widths. These random numbers are multiplied with appropriate width factor to get desired crack width at a given point on the zig-zag line. Hence numbers of zig-zag lines, width factor and scale factor are the parameters that can be controlled in the process of generation of the crack. Fig. 8.1(a) shows a surface random crack.

The image binarization technology is employed to define the crack location and concrete region on a 2D image. The default binary value of image is set to be 1 for the concrete region, whereas 0 is assigned to cracked part. The boundary between crack and solid phase can then be defined clearly.

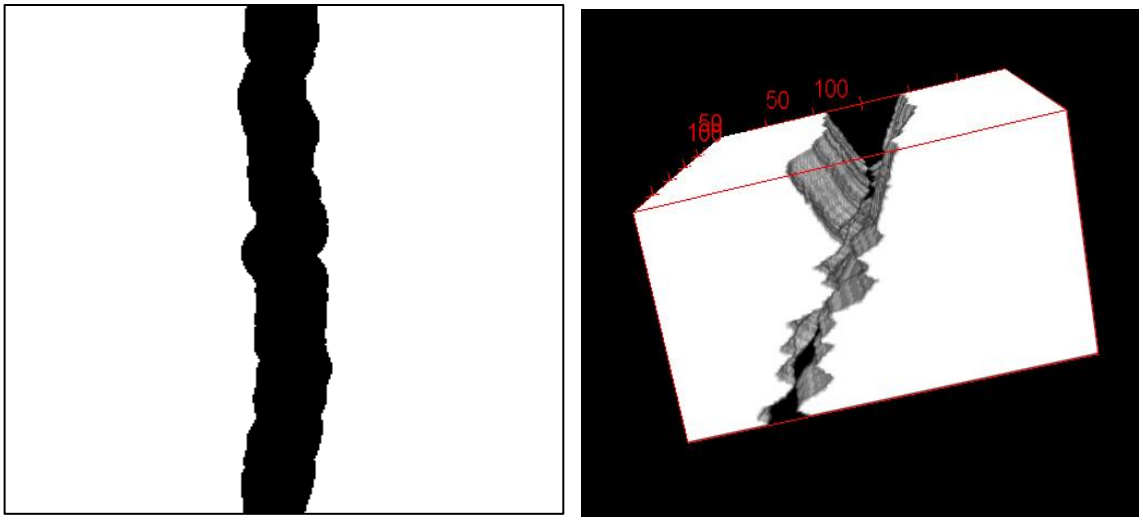


Figure 8.1. Generation of random 3D crack: (a). Random generated surface 2D crack; (b). 3D reconstruction of crack in a concrete block

8.2 3D Crack Generation

Based on the 2D image generated as explained in the previous section, these images were stacked together by establishing appropriate relationships between each layer to get the final 3D image. Since the minimum unit of the image is 1 pixel, each layer has a thickness of 1 pixel. Another set of random numbers have been generated to control the change in crack position between different layers. A linear function is used to control the width change. Finally a random crack can be generated as shown in Fig. 8.1(b). The random crack can be defined to pass through the entire member or can be terminated at any desired depth location in the 3D reconstruction of concrete.

9 SIMULATION RESULTS AND ANALYSIS

For several sets of 3D crack models, different mathematical models have been applied in this study to evaluate the influence of crack parameters on permeability. Geometric parameters are controlled during the generation process of crack. Results from both Navier-Stokes equation and Lattice-Boltzmann method are compared with Louis Equation in this section. It should be noticed that since crack width is varied everywhere on its path, it is difficult to determine the effective width of a crack and is not intuitive to describe the crack with this parameter alone as is commonly attempted. However, effective crack width has a linear relationship with the total volume of crack, which is more convenient to understand the relationship between the permeability and the crack geometry. Hence, the fraction of crack volume over the total concrete volume, defined as crack volume ratio (ϕ) in this study, rather than the effective crack width, is used as an important factor that influences the crack permeability.

$$\phi = \frac{V_c}{V_T} = \frac{w_e L_e d}{V_T} \tag{9-1}$$

Where w_e is the effective crack width, L_e is the effective crack length, d is the crack depth and V_T is the total volume of sample concrete. All of the parameters can be carefully controlled and quantified during the 3D crack generation process.

A parametric study was carried out to understand effect of various parameters such as effective crack width, crack volume, crack surface roughness, tortuosity on the permeability of the concrete. Effects of these different parameters are discussed separately in the following sub-sections.

9.1 Crack Volume Ratio (ϕ)

Fig. 9.1 shows a typical permeability – crack volume ratio relationship of crack with a tortuosity of 0.238 and global roughness of 22.54 μm. As shown in the curve, Louis equation results in larger permeability predictions than the Navier-Stokes equation. Since during crack width calculation, diameters of each circle are used to calculate the average crack width of each layer, which is likely to lead to a result that is higher than the actual value. For both the methods, it is obvious that the rate of permeability increase slows down drastically after ϕ reaches a threshold value of about 0.05. It should also be noted that the simplified Navier-Stokes equations cannot predict the crack permeability for crack volume ratio lower than 0.04. This is because during the numerical simulation, laminar flow is considered as the only flow going through the crack. However, when the effective crack width is decreased without a change in the pressure gradient, the velocity of flow will increase and turbulence will occur, which will mathematically invalidate the Navier-Stokes equations solution.

Fig. 9.2 shows the comparison of the permeability values from these above two methods. It can be found that for crack volume ratio between 0.05 and 0.14, a power function can fit the relationship well. With a lower crack volume ratio, the difference between these two methods drastically enlarges to more than one magnitude. This again gives the idea that for small effective width, the simplified Navier-Stokes equations cannot predict the permeability properly.

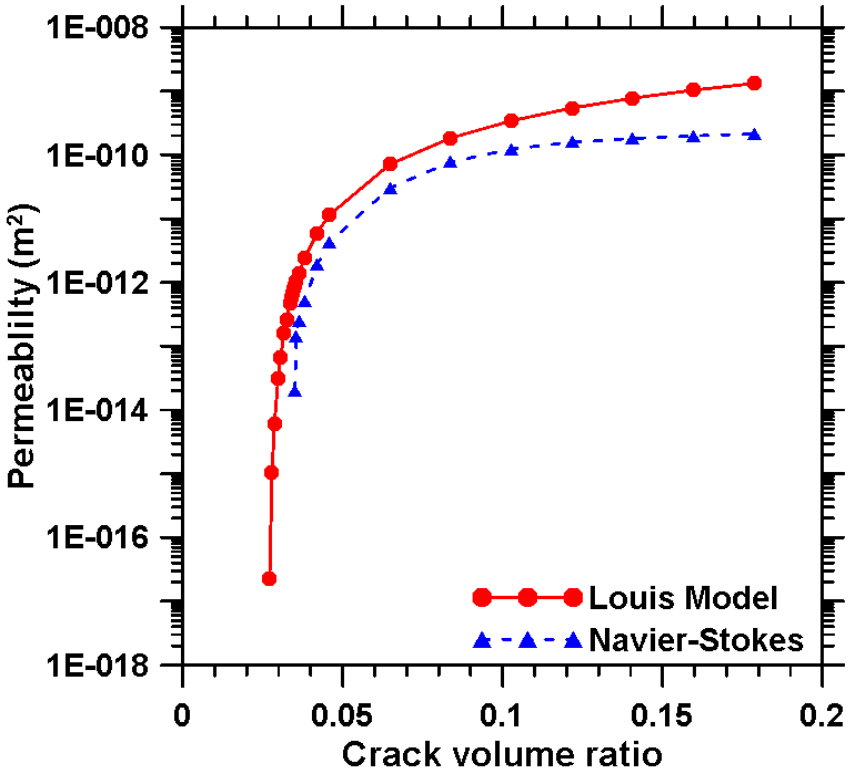


Figure 9.1. Permeability versus crack volume ratio, Navier-Stokes equations and Louis model

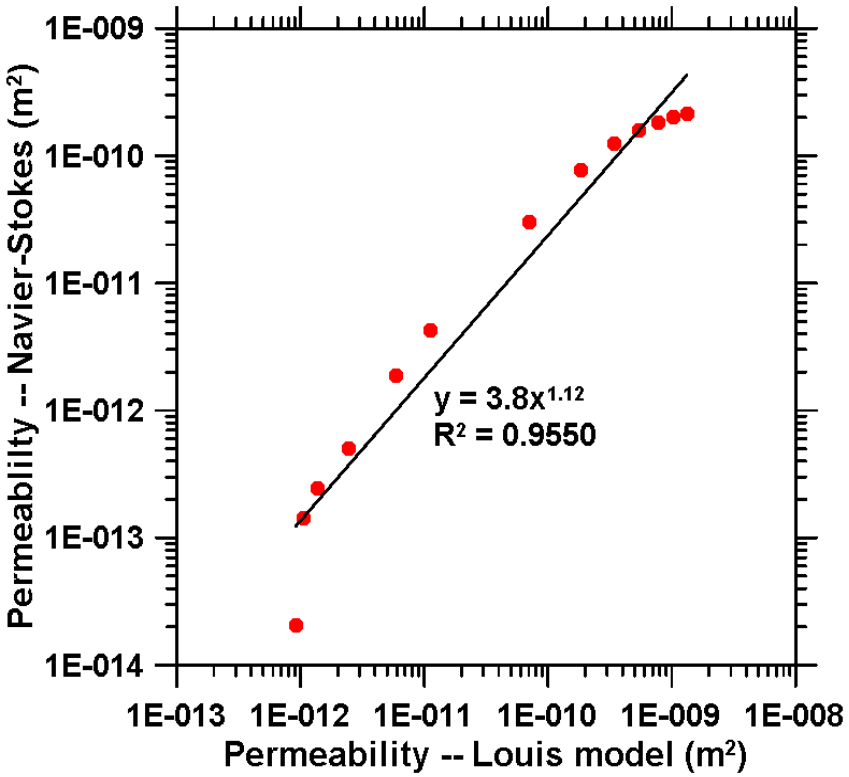


Figure 9.2. Comparison of Louis and Navier-Stokes prediction of permeability

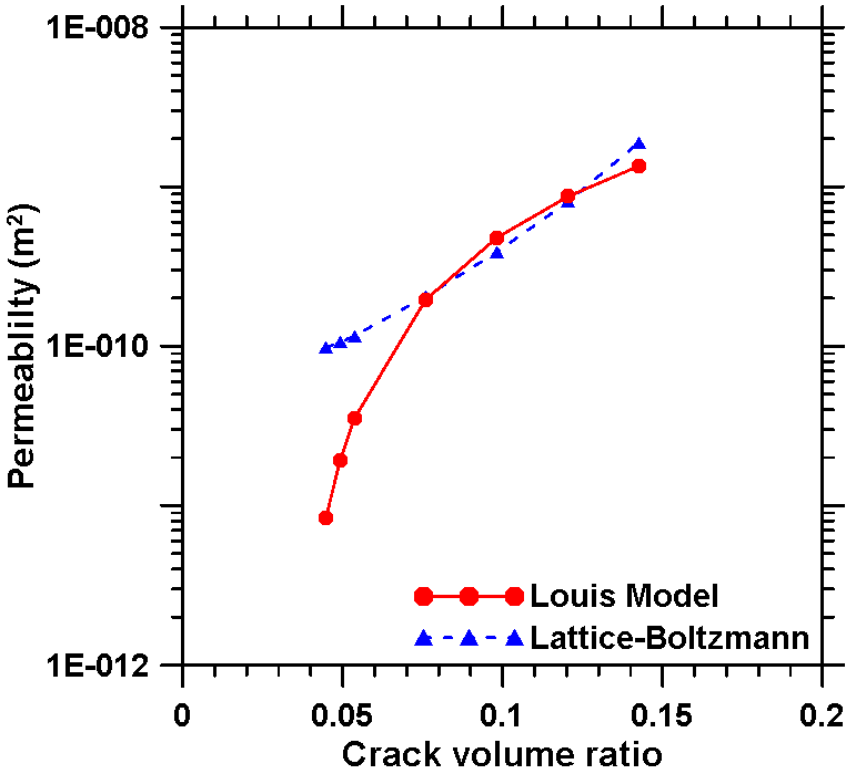


Figure 9.3. Permeability – crack volume ratio relationship, Lattice Boltzmann method and Louis model

Fig. 9.3 shows the permeability – effective crack width relationship determined using the Louis' equation and the Lattice-Boltzmann method. As shown here, Louis' equation gives a result that is closer to that of the Lattice Boltzmann method at crack volume ratios between 0.075 and 0.13. Fig. 9.4 shows the comparison of the predictions from Lattice Boltzmann method and the Louis model. A better linear correlation is found and it shows that the Lattice Boltzmann method gives a permeability value that is about 1.2 times larger than that given by Louis' equation. It can therefore be noticed that the LB equation adequately captures the predictions by the Louis equation for the conditions simulated in this study.

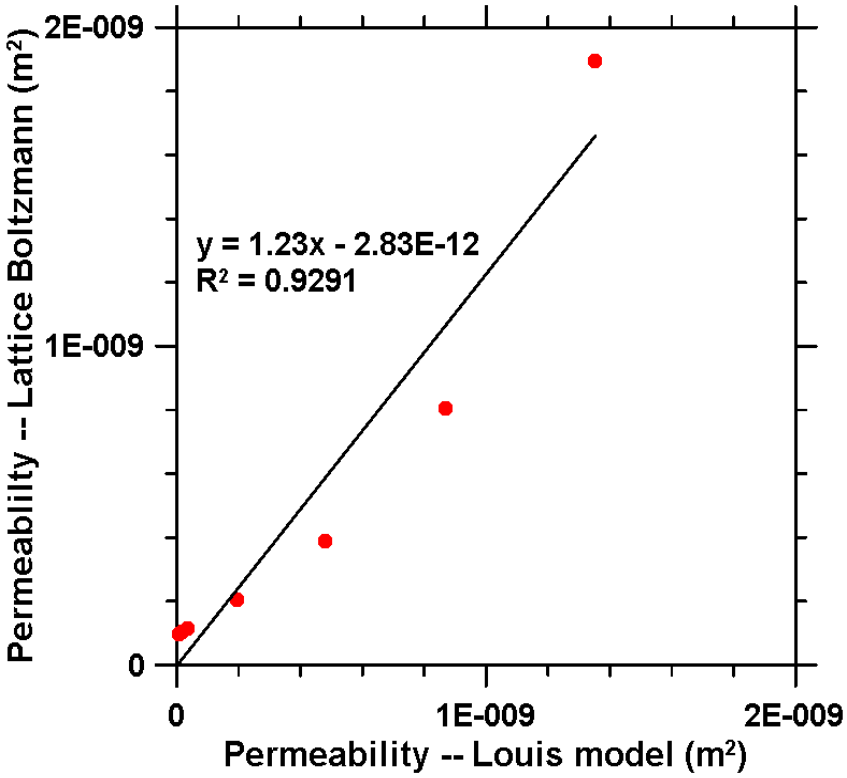


Figure 9.4. Comparison of Louis and Lattice-Boltzmann prediction of permeability

9.2 Effect of Crack Roughness and Tortuosity

Fig. 9.5 shows a typical curve for the relationship of global roughness and sampling length, λ, which gives an almost linear relationship between these two parameters. It indicates that the global roughness increases with an increase in sampling length. However it needs to be pointed out that the random numbers generated by the MATLAB code to control the perpendicular zig-zag shape also have a significant effect on roughness.

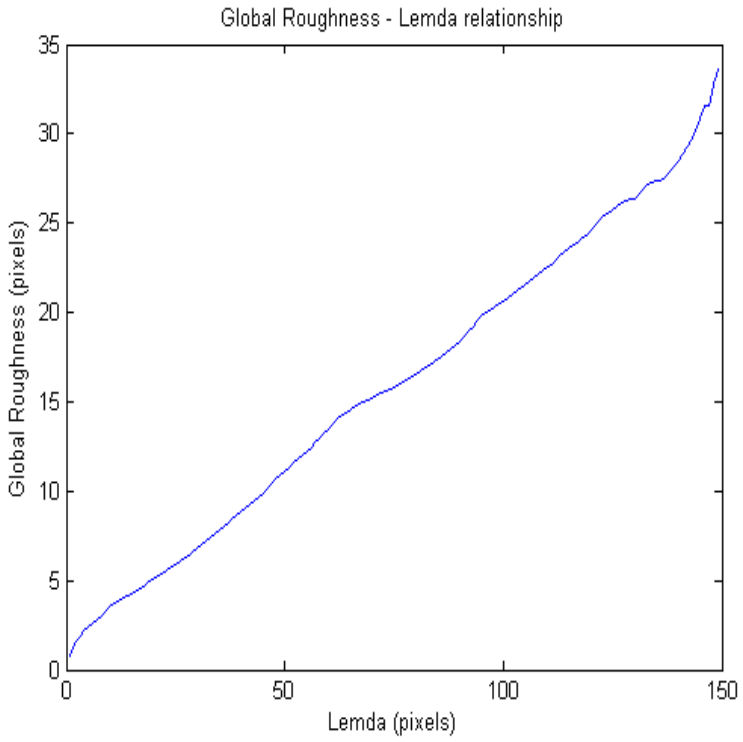


Figure 9.5. Relationship between roughness and sampling length

Fig. 9.6 shows a series of cracks with different boundary conditions: from smooth surface to rough surface. Table 9.1 shows the roughness values for a certain sampling length ($\lambda = 5$ pixels). It is obvious that with a significant change in crack boundary conditions, the values of roughness changes barely with a small sampling length. Thus, for this roughness measurement method, sampling length has a more significant effect on roughness. In other words, the value of surface roughness mainly depends on the sampling length, not the model itself. This needs to be considered while these models are being implemented, which is an objective of further work.

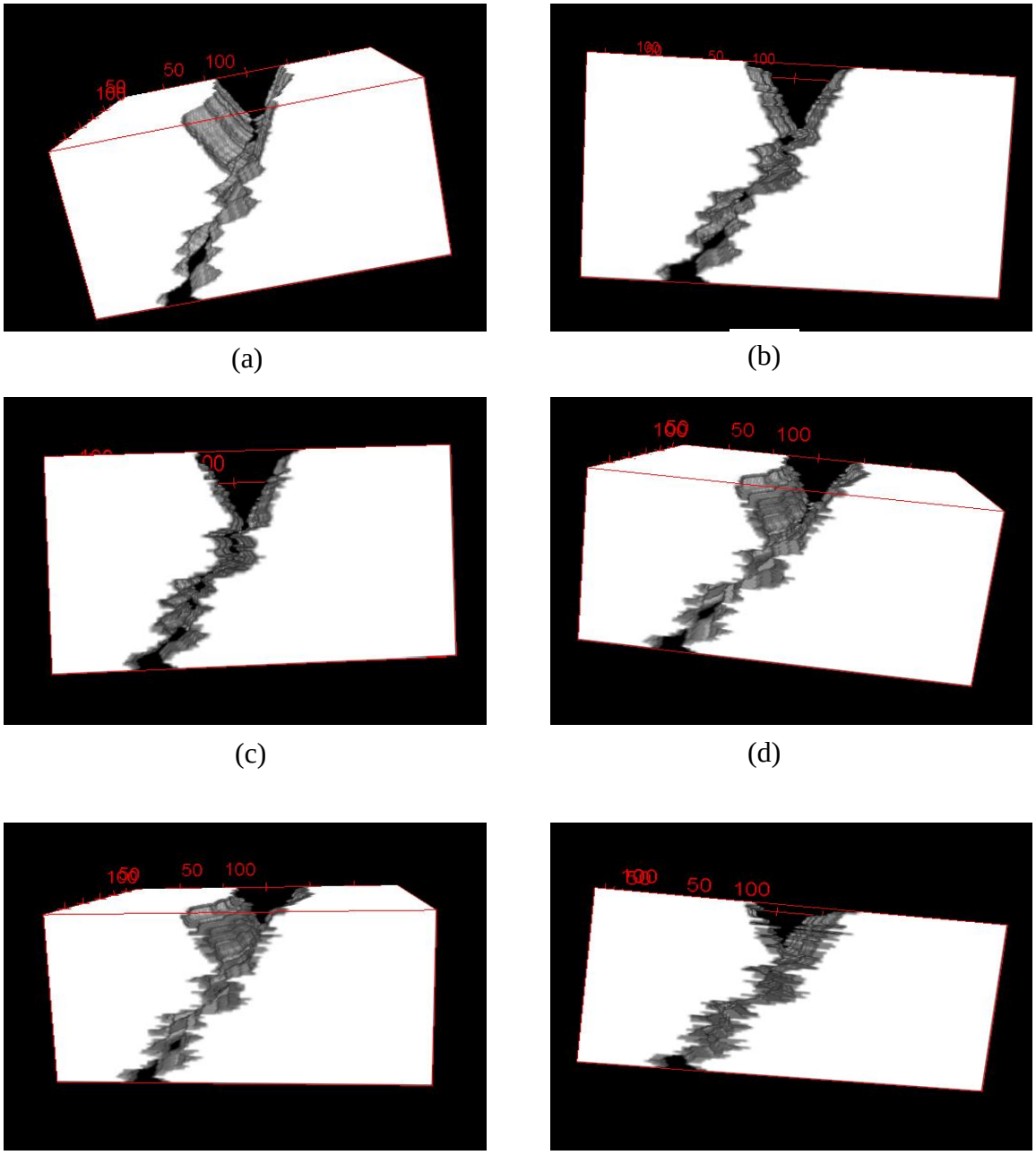


Figure 9.6. Different crack boundary conditions

Table 9.1 Roughness values with $\lambda = 5$ pixels

Conditions	(a)	(b)	(c)	(d)	(e)	(f)
Values(pixels)	10.59331	11.86364	13.54534	15.24518	16.92493	18.61767

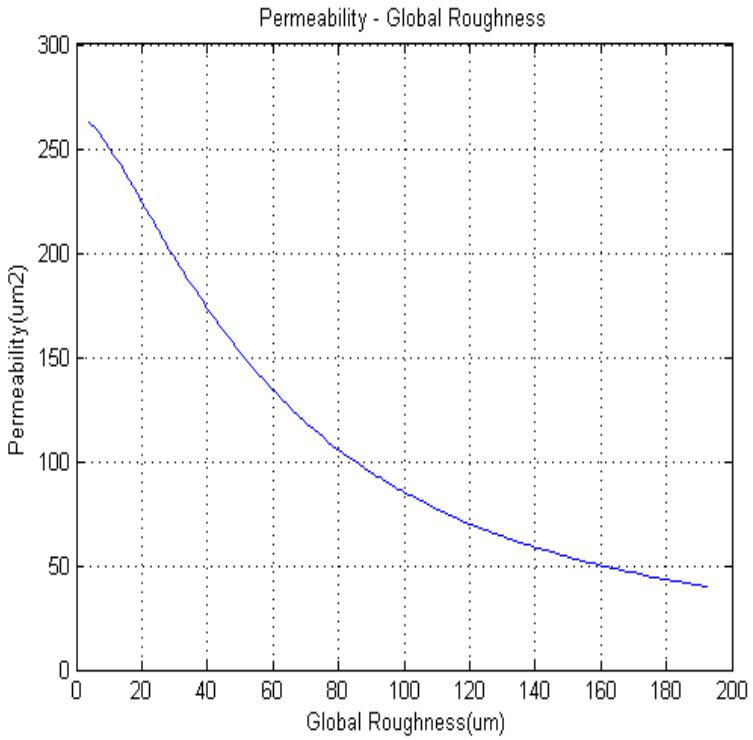


Figure 9.7. Permeability – Global roughness relationship for Louis model

Fig. 9.7 shows a typical permeability – roughness curve based on the Louis' model. This relationship shows that with an increasing global roughness, the permeability drops quickly since large roughness contributes to a larger friction coefficient, which causes an increase in pressure loss during the flow of the fluid through the crack.

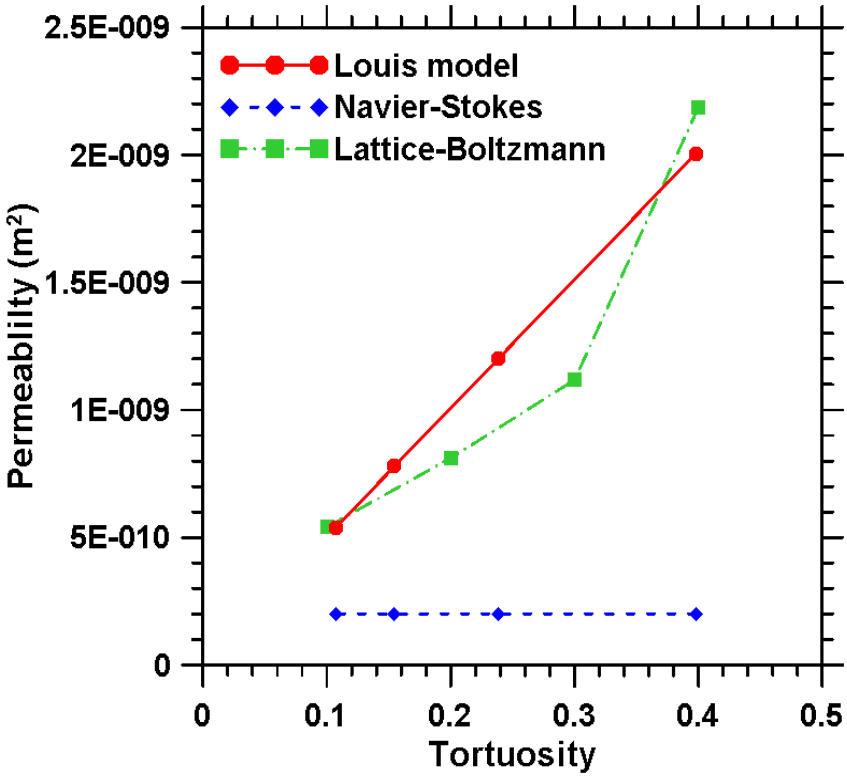


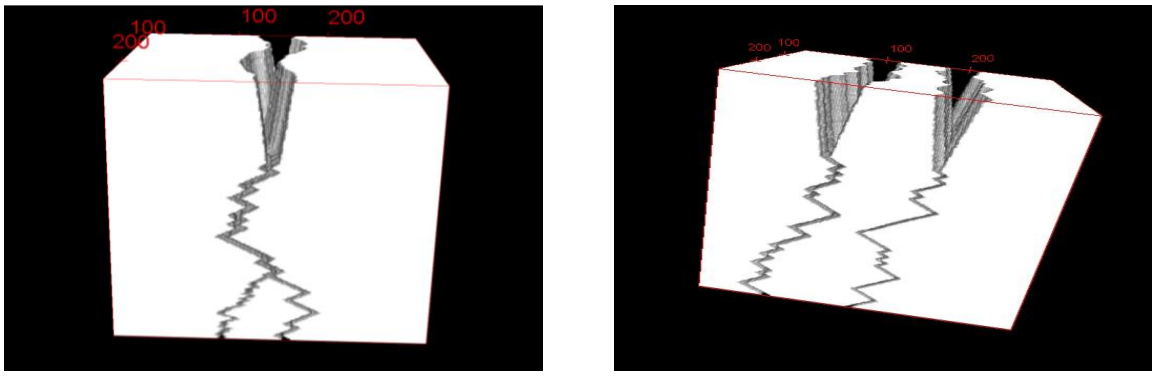
Figure 9.8. Permeability – Tortuosity relationship

Fig. 9.8 shows the permeability – tortuosity relationships for all the three methods. It is clear that Louis model gives a linear relationship between permeability and tortuosity since it is a linear function of tortuosity. However it will be obvious to the reader when considering the nature of tortuous pathways through a material, that such a relationship is not very practical. For the prediction based on Navier-Stokes' model, the results are quite invariant with tortuosity, which is not a realistic scenario. Several issues might have resulted in such an observation – the turbulence in transport pathways which are not considered by the model, and the simplifications in fluid velocity profiles. However, the Lattice-Boltzmann method is seen to predict the expected influence of tortuosity on the permeability of a cracked system. This is attributable to the refinements in this method that accurately predicts the velocity vectors and the solution of the incompressible flow equation.

9.3 System with Multiple Cracks

Compared to a system with a single crack as discussed above, multiple cracks are more common in practice. With more than one crack generated in a cubic sample, permeability will depend also on the crack connectivity (or the lack of it). .

In this part of the study, cracks with branching and a separate-double-crack (Fig. 9.9) are two basic models that will be discussed. The total crack volume ratio φ and the crack tortuosities (of the individual crack) are kept as a constant during the test. By varying the tortuosity and effective width of the crack, differences between single, double and branching cracks are discussed in the following sections.



(a) (b)

Figure 9.9. Multiple cracks with different connection: (a). Crack with branching; (b). Double-crack

Fig. 9.10 provides a comparison of the permeability values obtained from the Louis' equation and the Lattice Boltzmann method for single, double, and branching cracks with a crack volume ratio of 0.1 and a tortuosity of 0.3. The results show that when the Lattice Boltzmann method is used, despite the geometric shape of crack, a similar total volume of crack is found to lead to similar permeability values. It also should be noted that the double crack and branching crack geometries show exactly same permeability values, which is slightly lower than that of single crack. This phenomenon indicates that by

applying the Lattice Boltzmann method, multiple cracks lead to same permeability values regardless of their connectivity. However, for a certain total crack volume, increasing crack numbers reduces the concrete permeability. The Louis equation provides lower permeability values for both the double and branching cracks. This may indicate that this method is improper for calculating the permeability of multiple crack system. The preliminary studies shown here with multiple cracked systems indicate that both the methods, as applied here, are incapable of predicting the permeability through the crack. These could be related to the limitations in the Lattice Boltzmann method where flow paths are maintained irrespective of the connectivity when the sizes of the cracks are greater than a certain threshold value. More studies are needed to refine these models to account for the crack characteristics in systems with multiple cracks.

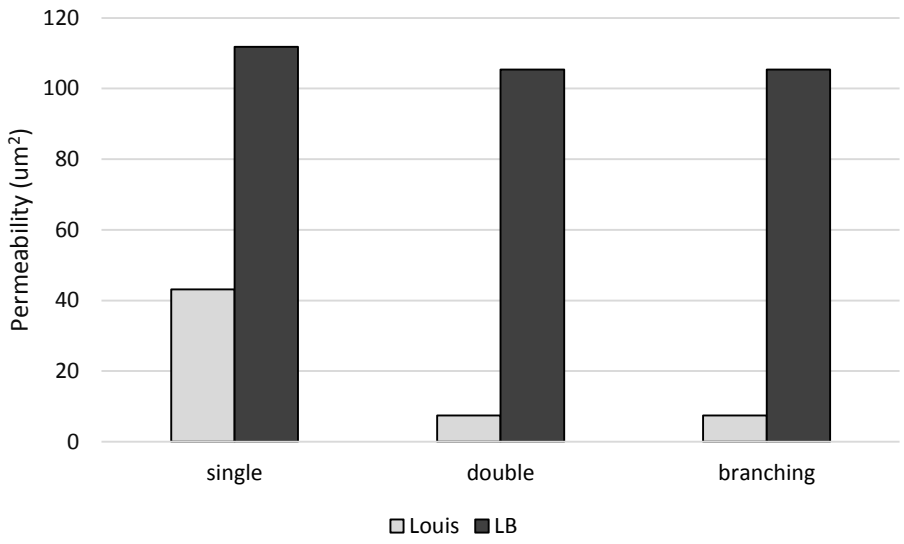


Figure 9.10. Permeability prediction from Lattice-Boltzmann and Louis model for different types of cracks

10 SUMMARY

In this part of the work, a method to develop 3D reconstruction of cracks is established. The major geometric properties of the crack (volume ratio, tortuosity, and the type of crack) are used as the input information to control the process of generation cracks. Other geometrical properties like crack roughness are measured during the generation process. The 3D crack generation models are used as inputs in both Navier-Stokes equation and a Lattice Boltzmann method for predicting the permeability of simulated cracked concretes. Louis' equation estimates the permeability based on all the geometrical properties of cracks and the empirical equation that relates those properties. Results on comparing the efficiency of these methods show that:

1) Both Navier-Stokes equations and Lattice Boltzmann method are not sensitive to the global surface roughness of crack
2) Lattice Boltzmann method and the Louis equation predicts the permeability better than the Navier-Stokes equation
3) Crack volume ratio and tortuosity are two dominant factors that influence the permeability of cracks
4) The studied methods do not adequately capture the permeability of concretes with multiple crack geometries and differing connectivity.

CONCLUSIONS

Chloride transport into concrete and crack of concrete are two important aspects that will drastically influence the durability of concrete. The first part of the thesis gives fundamental information on the transport of chloride ions into concrete under electrically induced conditions by considering the simultaneous movement of other ions in the concrete pore solution. By simulating the non-steady state migration test, the ion transportprocess is explored for OPC and modified concrete systems. The governing equation of the system is a convection-diffusion equation, and the accuracy of its solution is heavily dependent on both the element number and time step used in simulation. In this study, a total of 1000 elements and a time step of one second are chosen as default values for the models. While decreasing the numbers of elements leads to oscillatory results, increasing the value of time step results in unstable results. By using the Petrov-Garlerkin method, unconditional stable solution can be reached without significantly losing accuracy. All the numerical simulation codes are included in Appendix A. When a strong external electrical voltage exists in the system, the diffusion component can be omitted during simulation since the electrical migration dominates the penetration process. However, the specific relationship between diffusion and electrical migration is still unclear. Further work is needed to explore how electrical field influence the ionic diffusion process. During the NSSM test, the ionic diffusion coefficients varies with the ionic concentration, and a higher concentration level gives a slower diffusion speed. Porosity, pore connectivity and the chloride binding ability of the binder are considered in the numerical model that adequately predicts chloride transport into concretes in this study. Larger porosity and pore connectivity, and a smaller chloride binding ability lead

to higher penetration depth. It is also noticed that the assumption of constant electrical field is improper. During the NSSM, electrical field varies with the ionic concentration. It should be pointed out that the predicted concentration profile differed from the experimentally obtained values even though the overall depths were similar, which also needs further evaluations.

Second part of this thesis focused on numerically simulating the permeability of concrete with cracks. 3D crack models of different shapes have been established computationally. A MATLAB code for generating 3D cracks can be found in Appendix B. The Navier-Stokes equation and an 8-node Lattice Boltzmann method are applied to predict the permeability of the simulated cracked concrete along with the empirical Louis' equation that considers crack parameters. Numerical results show that: 1) Both Navier-Stokes equations and Lattice Boltzmann method are not sensitive to the global surface roughness of crack; 2) Lattice Boltzmann method and the Louis equation predicts the permeability better than the Navier-Stoke's equation; 3) Crack volume ratio and tortuosity are two dominant factors that influence the permeability of cracks; and 4) The studied methods do not adequately capture the permeability of concretes with multiple crack geometries and differing connectivity. MATLAB code for Lattice Boltzmann method is attached in Appendix C. More studies are needed to develop adequate numerical solutions for permeability in cracked concretes.

REFERENCES

[1] M.V.A Marinescu and H.J.H Brouwers, Free and bound chloride contents in Cementitious materials, 8th fib PhD Symposium in Kgs.Lyngby, Denmark, June 20-23, 2010

[2] Ming-Te Liang, Ran Huang and Hao-Yuan Jheng, Revisited to the relationship between the free and total chloride diffusivity in concrete, Journal of Marine Science and Technology, Vol. 18, 2010, pp. 442-448

[3] L. Tang, Chloride transport in concrete – measurement and prediction, PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 1996

[4] H. Zabara, Binding of external chlorides by cement pastes, PhD thesis, University of Toronto, Canada, 2001

[5] Qiang Yuan, Caijun Shi, Geert De Schutter, Dehua Deng and Fuqiang He, Numerical model for chloride penetration into saturated concrete, Journal of Materials in Civil Engineering, Vol. 23, 2011, pp.305-311

[6] G.A. Julio-Betancourt, R.D. Hooton, Study of the Joule effect on rapid chloride permeability values and evaluation of related electrical properties of concretes, Cement and Concrete Research, Vol. 34, 2004, pp. 1007–1015

[7] NT BUILD 492, Concrete, mortar and cement-based repair materials: chloride migration coefficient from non-steady state migration experiments, Nordtest Method, 492, 1999

[8] A. Matthew, Mechanical and chloride transport performance of particle size classified limestone cements, MS thesis, Arizona State University, U.S.A, 2014

[9] E.Samson, G.Lemaire, G.Marchand and J.Beaudoin, Modeling the chemical activity in strong ionic solutions, Computational Material Science, Vol. 15, 1999, pp. 285–294

[10] O.Truc, Prediction of chloride penetration into saturated concrete – Multi-species approach, PhD. Thesis, Deptartment of Building Materials, Chalmers University of Technology, Goteborg, Sweden

[11] E. Samson, J. Marchand, J.L. Robert, J.P. Bournazel, Modelling ion diffusion mechanisms in porous media, International Journal for Numerical Methods in Engineering, Vol. 46, 1999, pp. 2043-2060

[12] E. Samson, J. Marchand, Numerical solution of the Extended Nernst-Planck model, Journal of Colloid and Interface Science, Vol. 215, 1999, pp. 1-8

[13] O.C. Zienkiewicz, R.L. Taylor, The finite element method, 5th edition, Vol. 3, 2000, ISBN 0-7506-5050-8

[14] H.F.W. Taylor, A method for predicting alkali ion concentrations in cement pore solutions, Advanced Cement Research, Vol. 1, 1987, pp. 5–16

[15] W. Chen, Z.H. Shui, H.J.H. Brouwers, A computed-based model for the alkali concentrations in pore solution of hydrating Portland cement paste, Excellence in Concrete Construction through Innovation – Limbachiya & Kew (eds), © 2009 Taylor & Francis Group, London, ISBN 978-0-415-47592-1

[16] E.Samson, J. Marchand and K.A.Snyder, Calculation of ionic diffusion coefficients on the basis of migration test results, Materials and Structures, Vol. 36, 2003, pp. 156-165

[17] N. Neithalath, J. Weiss and J. Olek, Modeling the effects of pore structure on the acoustic absorption of Enhanced Porosity Concrete, Journal of Advanced Concrete Technology, Japan Concrete Institute, Vol.3, 2005, pp. 29-40

[18] K.B. Sanish, N. Neithalath and M. Santhanam, Monitoring the evolution of material structure in cement pastes and concretes using electrical property measurements, Constructing and Building Materials. Vol. 49, 2013, pp. 288–297

[19] E.J. Garboczi, Permeability, diffusivity, and microstructural parameters: A critical review, Cement and Concrete Research, Vol. 20, 1990, pp. 591–601

[20] K.A. Snyder, X. Feng, B.D. Keen, T.O. Mason, Estimating the conductivity of cement paste pore solutions from OH^-, K^+ and Na^+ concentrations, Cement and Concrete Research, Vol. 33, 2003, pp. 793–798

[21] P. Spiesz, H.J.H. Brouwers, The apparent and effective chloride migration coefficients obtained in migration tests, Cement and concrete Research, Vol. 48, 2013, pp.116-127

[22] Q. Yuan, C. Shi, G.D. Schutter, K. Audenaert, D. Deng, Chloride binding of cement based materials subjected to external chloride environment – a review, Construction and Building Materials, Vol. 23, 2009, pp. 1–13

[23] R. Loser, B. Lothenbach, A. Leemann, M. Tuchschmid, Chloride resistance of concrete and its binding capacity – comparison between experiments and thermodynamic modeling, Cement and Concrete Composite, Vol. 32, 2010, pp. 31–42

[24] M. Balonis, B. Lothenbach, G.L. Saout, F.P. Glasser, Impact of chloride on the mineralogy of hydrated Portland cement systems, Cement and Concrete Research, Vol. 40, 2010, pp. 1009-1022

[25] P. Spiesz, M.M. Ballari, H.J.H. Brouwers, RCM: A new model accounting for the non-linear chloride binding isotherm and the non-equilibrium conditions between the free- and bound-chloride concentrations, Construction and Building Materials, Vol. 27, 2012, pp. 293-304

[26] M. Castellote, C. Andrade, C. Alonso, Chloride-binding isotherms in concrete submitted to non-steady-state migration experiments, Cement and Concrete Research, Vol. 29, 1999, pp. 1799-1806

[27] J.L. Marriaga, P. Claisse, Determination of the concrete chloride diffusion coefficient based on an electrochemical test and an optimization model, Materials Chemistry and Physics, Vol. 117, 2009, pp. 536-543

[28] A. Ipavec, T. Vuk, R. Gabrovsek, V. Kaucic, Chloride binding into hydrated blended cements: The influence of limestone and alkalinity, Cement and Concrete Research, Vol. 48, 2013, pp. 74-85

[29] J. Marchand, B. Gérard, A. Delagrave, Ion transport mechanisms in cement-based materials, Materials Science of Concrete, Vol. 5, 1998, pp. 307-400

[30] S. Mindess, J.F. Young, and D. Darwin, Concrete, 2nd edition, Prentice Hall, Upper Saddle River, NJ, 2003

[31] A.S. EI-Dieb, R.D. Hooton, Water-permeability measurement of high performance concrete using a high-pressure triaxial cell, Cement and Concrete Research, Vol. 25, 1995, pp. 1199-1208

[32] D. Ludirdja, R.L. Berger, J.F. Young, Simple method for measuring water permeability of concrete, ACI Materials Journal, Vol. 86, 1989, pp. 433–439

[33] A.J. Katz, A.H. Thompson, Quantitative prediction of permeability in porous rock, Physical Review B, Vol. 34, 1986, pp. 8179–8181

[34] P. Halamickova, R.J. Detwiler, D.P. Bentz, E.J. Garboczi, Water permeability and chloride ion diffusion in portland cement mortars: relationship to sand content and critical pore diameter, Cement and Concrete Research, Vol. 25, 1995, pp. 790–802

[35] M.R. Nokken, R.D. Hooton, Using pore parameters to estimate permeability or conductivity of concrete, Materials and Structures Journal, Vol. 41, 2008, pp. 1–16

[36] A. Kermani, Permeability of stressed concrete, Building Research Information, Vol. 19, 1991, pp. 360–366

[37] M. Tsukamoto, J.D. Wörner, Permeability of cracked fiber-reinforced concrete, Darmstadt Concrete, Vol. 6, 1991, pp. 123–135

[38] B. Gérard, D. Breysse, A. Ammouche, O. Houdusse, O. Didry, Cracking and permeability of concrete under tension, Materials and Structures, Vol. 29, 1996, pp. 141–151

[39] A. Akhavan, S.M.H. Shafaatian, F. Rajabipour, Quantifying the effects of crack width, tortuosity and roughness on water permeability of cracked mortars, Cement and Concrete Research, Vol. 42, 2012, pp. 313-320

[40] D.Snow, Anisotropic permeability of fractured media, Water Resource Research, Vol. 5, 1969, pp. 1273-1289

[41] J. Bear, Dynamics of fluids in porous media, Dover Publications, New York, 1988

[42] Avizo 7 users guide, 2011, Visualization Dciences Group

[43] S. Chen, G.D. Doolen, Lattice Boltzmann method for fluid flows, Annual Review of Fluid Mechanism, Vol. 30, 1998, pp. 329-364

[44] B.Chopard, M. Droz, Cellular Automata Modeling of Physical Systems, Claude Godrèche (ed.): Collection Aléa-Saclay: monographs and texts in statistical physics, 1998, Cambridge: Cambridge University Press

[45] D. Kandhai, D.J.E. Vidal, A.G. Hoekstra, H. Hoefsloot, P. Iedema, P.M.A. Sloot, Lattice-boltzmann and finite element simulations of fluid flow in a SMRX static mixer reactor, International Journal for Numerical Methods in Fluids, Vol. 31, 1998, pp. 1019-1033

[46] S. Succi, The lattice Boltzmann Equation: for Fluid Dynamics and Beyond, Numerical Mathematics and Scientific Computation, 1998, Oxford New York: Oxford University Press

[47] P.L. Bhatnagar, E.P. Gross, M. Krook, A model for collision processes in gases: I. Small amplitude processes in charged and neutral one-component systems, Physics Review, Vol. 94, 1954, pp. 511-525

[48] S. Ansumali, I.V. Karlin, E. Frouzakis, K.B. Boulouchos, Entropic lattice Boltzmann method for microflows, Journal of Statistical Physics, Vol. 107, 2002, pp. 291-309

[49] Z. Guo, B. Shi, N. Wang, Lattice BGK Model for Incompressible Navier-Stokes Equation, Journal of Computational Physics, Vol. 165, 2000, pp. 288-306

APPENDIX A

C^{++} CODE FOR NUMERICAL SIMULATION OF NSSM TEST

1. Main program

```
/*This is a project for 1D diffusion simulation
based on NPP equation in saturated concrete
Author: Pu Yang
Date: Feb. 2014
*/

#include<iostream>
#include"mesh.h"

int main()
{
	CMesh DIFFUSION;

	// read & prepare the model
	DIFFUSION.ReadProblem1();

	DIFFUSION.PrepareModel();

	DIFFUSION.ReadProblem2();

	DIFFUSION.PrepareIO();

	// impose bondary and initial conditions
	DIFFUSION.ConstructIC();

	// construct the capacity matrix
	DIFFUSION.ConstructC();

	// construct the force vector
	DIFFUSION.ConstructF();

	int Step; int total; int kind;
	DIFFUSION.GetStep(Step);
	DIFFUSION.GetTotal(total);
	DIFFUSION.Getkinds(kind);

	while (Step < total)
	{
		std::cout<<"Total Steps: "<<Step<<"\n";

		DIFFUSION.Counter();

		DIFFUSION.GetStep(Step);

		DIFFUSION.Solve();

		for (int i=1;i<=kind;i++)
		{
			DIFFUSION.Solve2(i);
		}

		DIFFUSION.Solve3();

	}
	DIFFUSION.ShowEnd ();
```

```
	return 0;
}
```

2. Header file

```
#ifndef _MESH_H__
#define _MESH_H__

#include <fstream>
#include <iostream>
#include <sstream>
using std::ostringstream;
#include "node.h"                       // node class
#include "element.h"                    // element class
#include "constants.h"                  // program limitations etc.
#include "..\library\vectortemplate.h"
#include "..\library\matrixtemplate.h"
#include "..\library\MatToolBox.h"
#include "..\library\NumericalIntegration.h"

class CMesh
{
public:
	CMesh ();  // ctor
	~CMesh (); // dtor



	 // study case
	 void Model1 ();void Model2 ();void Model3 ();void Model4 ();
	 void Model5 ();void Model6 ();void Model7 ();

	 void BD2 ();


	 // help function
	 void IO ();
	 void PrepareIO ();
	 void ReadProblem1 ();
	 void ReadProblem2 ();
	 void PrepareModel ();
	 void PrepareMode2 ();
	 void ShowEnd ();
	 void ConstructK_c ();
	 void ConstructK_t ();
	 void ConstructC ();
	 void ConstructCcl ();
	 void ConstructIC ();
	 void ConstructE ();
	 void ConstructBC ();
	 void ConstructF ();
	 void ImposeBC ();
	 void CalculateCl ();
	 void CalculateD ();
	 void Solve ();
	 void Solve2 (int i);
```

```
        void Solve3 ();
        void Solve4 ();
        void Counter ();
        int CreateOutputCon (int& j,CMatrix<double>& A, int& T);
        int CreateOutputFlux (int& j,CMatrix<double>& A, int& T);
        int CreateOutputCol (int& j,CMatrix<double>& A, int& T);
        int CreateOutputCon2 (int j,CVector<double>& A, int& T);
        int CreateOutputCol2 (int j,CVector<double>& A, int& T);

        int CreateOutput2 (int j,CVector<double>& A, int& T);
        int WriteReport2 (CVector<double>& A);
        int WriteReport3 (CMatrix<double>& A);
        int WriteReport ();
        void GetStep (int& step);
        void GetTotal (int& total);
        void GetTime (float& t);
        void Getkinds (int& kind);


        //void Crout(int d,CVector<double>S,CVector<double>D);
        void solveCrout(int d,double*LU,double*b,double*x);

private:
       int m_nNodes;         // number of nodes
       int m_nElements;      // number of element
       int m_nKinds;         // total number of ions
       double m_dLength;     // thichness of sample
       double m_dArea;       // section area of smaple
       double m_fdistance;   // distance between two nodes
       double m_fdt;          // time step
       int m_nStep;          // total step
       int N;                     // current step
       int TIME;                // current time
       double m_dp;            // porosity
       double m_dtau;          // tortuosity
       double m_da;            // constant a
       double m_db;            // constant b
       double m_dden;          // density
       double m_dPhi;          // electrical potential
       double m_E;            // electrical potential gradient
       double m_D0;           // max cofficient
       double m_C0;           // max concentration
       double Pec;             // Peclet number
       double a;               // constant for binding mechanism
       double b;               // constant for binding mechanism
       double m_Cl;            // total chloride content in the sample
       int JUDGE;              // weither calculate binding or not, 1 yes 0 for no;



      // these store the FE model data
    CVector<CNode>            m_NodalData;          // nodal data
    CVector<CElement>         m_ElementData;        // element data


    CVector<ostringstream> m_FileInput;  // File Input
    CVector<ostringstream> m_FileOutput; // File Output
```

```
    std::ofstream m_FileOutputhandle[10];

       CVector<double> m_dC_s;      // surface concentration of different ions
       CVector<double> m_dC_0;      // bottom concentration of different ions
       CVector<double> m_dC_i;      // initial concentration of different ions
       CVector<double> m_nZ;        // velance of chloride iron
       CVector<double> m_dD;        // diffusion cofficient of different ions
       CVector<double> m_dE;              // electrical potential at each node
       CVector<double> m_SF;        // structural nodal force
       CVector<double> m_I;         // current at each node
       CVector<double> m_Phi;       // electrical potential at each node
       CMatrix<double> m_SSM1;      // structural stiffness matrix, constant part
       CMatrix<double> m_SSM2;      // structural stiffness matrix, variable part
       CMatrix<double> m_SSM3;      // structural stiffness matrix, variable part
       CMatrix<double> m_SSM4;      // structural stiffness matrix, variable part
       CMatrix<double> m_Ke;        // structural electric stiffness matrix
       CMatrix<double> m_SSC;       // structural capacity matrix
       CMatrix<double> m_J;         // flux of ions
       CMatrix<double> m_Coulombs;  // coulombs at each node
       CVector<double> coulombs;    // arveage coulombs
       CMatrix<double> m_SA0;       // "a" vector for initial condition of
different ions
       CVector<double> m_U;           // system vector
       CVector<double> m_Ccl;       // binding chloride at each node



       CMatrix<double> Kt;
       CVector<double> U_new;
       CVector<double> U_old;
       CVector<double> J_old;
       CMatrix<double> a_old;

       CMatrix<double> KK;
       CVector<double> FF;


       CMatrix<double> D;            // Diffusion matrix


       // element-related
    int k1DC0LElement (int nE, CVector<int>& nVNList,
                           CMatrix<double>& dMk);


};
#endif
```

3. Input file

```
#include"mesh.h"
#include<cmath>

void CMesh::BD2 ()
{
       // Boundary Condition
```

```
        // Cl- ion
        m_dC_s(1) = 1900;
        m_dC_0(1) = 0;
        // Na+ ion
        m_dC_s(2) = 1900;
        m_dC_0(2) = 300;
        // K+ ion
        m_dC_s(3) = 0;
        m_dC_0(3) = 0;
        // OH- ion
        m_dC_s(4) = 0.0;
        m_dC_0(4) = 300;
}



void CMesh::Model1 ()
{
        a = 0.48e-4;
        b = 1;
        m_dp = 0.099;
        m_dtau = 0.002789;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0;
        m_nZ(1) = -1;
        m_dD(1) = 0.211e-12;
        // Na+
        m_dC_i(2) = 140;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.211e-12;
        // K+
        m_dC_i(3) = 210;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.211e-12;
        // OH-
        m_dC_i(4) = 349.99;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.211e-12;
}

void CMesh::Model2 ()
{
        a = 0.37e-4;
        b = 1;
        m_dp = 0.122;
        m_dtau = 0.003681;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.278e-12;
        // Na+
        m_dC_i(2) = 110;
```

```
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.278e-12;
        // K+
        m_dC_i(3) = 160;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.278e-12;
        // OH-
        m_dC_i(4) = 269.999;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.278e-12;
}

void CMesh::Model3 ()
{
        a = 0.51e-4;
        b = 0;
        m_dp = 0.104;
        m_dtau = 0.001049;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.0792e-12;
        // Na+
        m_dC_i(2) = 110;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.0792e-12;
        // K+
        m_dC_i(3) = 160;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.0792e-12;
        // OH-
        m_dC_i(4) = 269.999;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.0792e-12;
}

void CMesh::Model4 ()
{
        a = 0.35e-4;
        b = 1;
        m_dp = 0.132;
        m_dtau = 0.005333;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.403e-12;
        // Na+
        m_dC_i(2) = 80;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.403e-12;;
        // K+
        m_dC_i(3) = 120;
```

```
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.403e-12;;
        // OH-
        m_dC_i(4) = 199.999;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.403e-12;;
}

void CMesh::Model5 ()
{
        a = 0.48e-4;
        b = 1;
        m_dp = 0.129;
        m_dtau = 0.001816;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.137e-12;
        // Na+
        m_dC_i(2) = 90;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.137e-12;
        // K+
        m_dC_i(3) = 130;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.137e-12;
        // OH-
        m_dC_i(4) = 219.999;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.137e-12;
}

void CMesh::Model6 ()
{
        a = 0.6e-4;
        b = 1;
        m_dp = 0.127;
        m_dtau = 0.001740;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.131e-12;
        // Na+
        m_dC_i(2) = 300;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.131e-12;
        // K+
        m_dC_i(3) = 250;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.131e-12;
        // OH-
        m_dC_i(4) = 549.999;
```

```
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.131e-12;
}

void CMesh::Model7 ()
{
        a = 10e-4;
        b = 0.405;
        m_dp = 0.132;
        m_dtau = 0.001235;
        // Note: All the values are in the unit of mmol/L

        // Initial Condition
        // Cl-
        m_dC_i(1) = 0.001;
        m_nZ(1) = -1;
        m_dD(1) = 0.0932e-12;
        // Na+
        m_dC_i(2) = 400;
        m_nZ(2) = 1;
        m_dD(2) = 1.334/2.032*0.0932e-12;
        // K+
        m_dC_i(3) = 270;
        m_nZ(3) = 1;
        m_dD(3) = 1.957/2.032*0.0932e-12;
        // OH-
        m_dC_i(4) = 669.999;
        m_nZ(4) = -1;
        m_dD(4) = 5.273/2.032*0.0932e-12;
    }
```

4. Functions file

Ed

```
#include"mesh.h"
#include<cmath>
#include<stdio.h>


CMatToolBox<double> MTB;


CMesh::CMesh()
{
        m_nNodes = 0;
        m_nElements = 0;
        m_nKinds = 0;
        m_dLength = 0;
        m_fdistance = 0;
        m_fdt = 0;
        m_nStep = 0;
        TIME = 0;
        N = 0;
        m_dp=0;
        m_da=0;
```

```
	m_db=0;
	m_dden = 0;
	m_dArea = 0;
	m_dPhi = 0;
	m_E = 0;
	m_C0 = 0;
	m_D0 = 0;
	Pec = 0;
	m_dtau = 0;
	a = 0;
	b = 0;
	m_Cl = 0;
	JUDGE = 1;
}

CMesh::~CMesh()
{
}


void CMesh::ReadProblem1 ()
{
	std::cout<<"Program starts: \n";
	m_nKinds = 4;
	m_nNodes = 100;
	m_nElements = m_nNodes - 1;
	m_dLength = 0.05;
	m_dArea = 0.1;
	m_dPhi = 30;

	m_dden = 2400;
	//double delta = 2.0f/(PI*PI)*(1/m_dD)*pow((m_dLength/m_nElements),2.0);
	//delta = floor(delta);
	m_nStep = 1200*24;
	m_fdt = 3;

	Pec = F/R/T*m_dPhi*0.5f/m_nElements;

	//std::cout<<delta<<"\n";
	std::cout<<" \nReadProblem: Done! \n";
}

void CMesh::PrepareModel ()
{
	m_fdistance = m_dLength/(m_nElements);
	m_E = m_dPhi/m_dLength;
	m_ElementData.SetSize(m_nElements);
	m_NodalData.SetSize(m_nNodes);
	m_SSM1.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
	m_SSM2.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
	m_SSM3.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
	m_SSM4.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
	m_Ke.SetSize(m_nNodes,m_nNodes);
	m_SSC.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
	m_SA0.SetSize(m_nKinds,m_nNodes);
	m_J.SetSize(m_nKinds,m_nNodes);
	m_Coulombs.SetSize(m_nKinds,m_nNodes);
	m_U.SetSize(m_nNodes*DOFPN);
```

```
m_dE.SetSize(m_nNodes);
m_SF.SetSize(m_nNodes*DOFPN);
m_I.SetSize(m_nNodes);
m_Phi.SetSize(m_nNodes);
m_dC_0.SetSize(m_nKinds);
m_dC_i.SetSize(m_nKinds);
m_dC_s.SetSize(m_nKinds);
m_nZ.SetSize(m_nKinds);
m_dD.SetSize(m_nKinds);
m_FileInput.SetSize(m_nKinds+4);
m_FileOutput.SetSize(m_nKinds+4);
m_SSM1.Set(0.0f);
m_SSM2.Set(0.0f);
m_SSM3.Set(0.0f);
m_SSM4.Set(0.0f);
m_Ke.Set(0.0f);
m_SSC.Set(0.0f);
m_SA0.Set(0.0f);
m_J.Set(0.0f);
m_Coulombs.Set(0.0f);
m_U.Set(0.0f);
m_dC_0.Set(0.0f);
m_dC_i.Set(0.0f);
m_dC_s.Set(0.0f);
m_nZ.Set(0.0f);
m_dD.Set(0.0f);
m_dE.Set(0.0f);
m_I.Set(0.0f);
m_Phi.Set(0.0f);
N = 0;
TIME = 0;

m_Ccl.SetSize(m_nNodes);
m_Ccl.Set(0.0f);

coulombs.SetSize(9);
coulombs.Set(0.0f);


Kt.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
Kt.Set(0.0f);
J_old.SetSize(m_nNodes);
J_old.Set(0.0f);
a_old.SetSize(m_nKinds, m_nNodes);
U_new.SetSize(m_nNodes*DOFPN);
U_new.Set(0.0f);
U_old.SetSize(m_nNodes*DOFPN);
U_old.Set(0.0f);

KK.SetSize(m_nNodes*DOFPN,m_nNodes*DOFPN);
KK.Set(0.0f);
FF.SetSize(m_nNodes*DOFPN);
FF.Set(0.0f);

D.SetSize(m_nKinds,m_nNodes);
D.Set(0.0f);

// set nodal coordinates
```

```
	for (int i=1;i<=m_nNodes;i++)
	{
		double fVC = (i-1)*m_fdistance;
		m_NodalData(i).SetCoords(fVC);
		//std::cout<<"Coordinate of "<<i<<"th nodal: "<<fVC<<"\n";
	}

	// set elements nodes
	for (int i=1;i<=m_nElements;i++)
	{
		CVector<int> nVNList(NUMENODES);
		nVNList(1) = i;
		nVNList(2) = i+1;
		m_ElementData(i).SetNodes(nVNList);
	}



	std::cout<<"\nPrepare model: Done! \n";

}

void CMesh::ReadProblem2 ()
{
	int m,n;
	std::cout<<"Please select the case you want to analysis:\n";
	std::cout<<"1:OPC, 2:0.2LS, 3:0.1LS+0.1MK, 4:0.35LS, 5:0.25LS+0.1MK,
6:0.2FA, 7:0.35FA\n"<<std::endl;
	std::cout<<"Please enter the case number: ";
	std::cin>>m;
	if (m==1)
		Model1();
	else if (m==2)
		Model2();
	else if (m==3)
		Model3();
	else if (m==4)
		Model4();
	else if (m==5)
		Model5();
	else if (m==6)
		Model6();
	else if (m==7)
		Model7();
	else
		std::cout<<"Wrong case number!";


	n = 2;

	if (n==2)
		BD2();
	else
		std::cout<<"Wrong case number!";

}
```

```
void CMesh::PrepareMode2 ()
{
	m_C0 = m_dC_s(1);
	m_D0 = m_dD(4);
	for (int i=1;i<=m_nKinds;i++)
	{
		m_dC_s(i) = m_dC_s(i)/m_C0;
		m_dC_i(i) = m_dC_i(i)/m_C0;
		m_dC_0(i) = m_dC_0(i)/m_C0;
		m_dD(i) = m_dD(i)/m_D0;
	}
	m_fdistance = m_fdistance/m_dLength;
	//m_dPhi = m_dPhi*F/R/T;
	m_fdt = m_fdt*m_D0/(m_dLength*m_dLength);
	m_E = m_dPhi/1;
}


void CMesh::ConstructIC ()
{
	for (int i=1;i<=m_nKinds;i++)
	{
		m_SA0(i,1) = m_dC_s(i);
		for (int j=2;j<m_nNodes;j++)
		{
			m_SA0(i,j) = m_dC_i(i);
		}
		m_SA0(i,m_nNodes) = m_dC_0(i);
		CreateOutputCon(i,m_SA0,TIME);
		for (int j=1;j<=m_nNodes;j++)
		{
			m_J(i,j) = 0;
		}
		CreateOutputFlux(i,m_J,TIME);
		CreateOutputCol(i,m_Coulombs,TIME);
	}
	for (int i=1;i<=m_nNodes;i++)
	{
		double Cl = 0;
		Cl = abs(m_SA0(1,i)/1000*35);
		m_Ccl(i) = a*pow(Cl,b);  // bound chloride, in g/g-solid
		m_I(i) =
F*(m_nZ(1)*m_J(1,i)+m_nZ(2)*m_J(2,i)+m_nZ(3)*m_J(3,i)+m_nZ(4)*m_J(4,i));
	}
	for (int i=2;i<=m_nNodes;i++)
	{
		m_Phi(i) =  m_Phi(i-1) + m_E*m_fdistance;
	}
	CreateOutput2(5,m_I,TIME);
	CreateOutput2(6,m_Phi,TIME);
	CreateOutputCon2(8,m_Ccl,TIME);

	for (int i=1; i<=m_nNodes; i++)
	{
		m_U(5*i-4) = m_SA0(1,i);
		m_U(5*i-3) = m_SA0(2,i);
		m_U(5*i-2) = m_SA0(3,i);
		m_U(5*i-1) = m_SA0(4,i);
```

```
		m_U(5*i)   = m_Phi(i);
	}

	std::cout<<"Construct IC is done!"<<std::endl;
}

void CMesh::ConstructE ()
{

	for (int i=1;i<=m_nNodes;i++)
	{
		m_dE(1) = (-3.0f*m_Phi(1)+4.0f*m_Phi(2)-m_Phi(3))/(2.0f*m_fdistance);
		for (int i=2;i<m_nNodes;i++)
		{
			m_dE(i) = (m_Phi(i+1)-m_Phi(i-1))/(2.0f*m_fdistance);
		}
		m_dE(m_nNodes) = (3.0f*m_Phi(m_nNodes)-4.0f*m_Phi(m_nNodes-
1)+m_Phi(m_nNodes-2))/(2.0f*m_fdistance);
	}

}

void CMesh::CalculateCl ()
{
	double x = 0, y = 0;
	for (int i=1;i<m_nNodes;i++)
	{
		x = 0.5*(m_SA0(1,i)+m_SA0(1,i+1))*m_fdistance;
		y += x;
	}
	m_Cl = y/m_dLength;

	x = m_dden*1.4/35;

	if (m_Cl>x)
	{
		ConstructCcl();
		JUDGE = 1;
	}
	//std::cout<<m_Cl<<"\n";
}

void CMesh::CalculateD()
{
	for (int i=1;i<=m_nNodes;i++)
	{
		if (m_SA0(1,i) >= 2000)
		{
			D(1,i) = m_dD(1);
			D(2,i) = m_dD(2);
			D(3,i) = m_dD(3);
			D(4,i) = m_dD(4);
		}
		else if (m_SA0(1,i) <= 10)
		{
			D(1,i) = m_dD(1)*pow(0.01,EXP);
			D(2,i) = m_dD(2)/m_dD(1)*D(1,i);
			D(3,i) = m_dD(3)/m_dD(1)*D(1,i);
```

```
                        D(4,i) = m_dD(4)/m_dD(1)*D(1,i);
                }
                else
                {
                        D(1,i) = m_dD(1)*pow((abs(m_SA0(1,i)/1000)),EXP);
                        D(2,i) = m_dD(2)/m_dD(1)*D(1,i);
                        D(3,i) = m_dD(3)/m_dD(1)*D(1,i);
                        D(4,i) = m_dD(4)/m_dD(1)*D(1,i);
                }
        }
}

void CMesh::ConstructK_t ()
{
        m_SSM4.Set(0.0f);
    int i, j, k;
    const int KSIZE = NUMENODES*DOFPN;      // size of the stiffness matrix
    CVector<int>    nVEDOF(KSIZE);          // dof associated with element
    CVector<int>    nVNList(NUMENODES);     // list of element nodes
       CMatrix<double> dMk4("k4",KSIZE,KSIZE);

       double E = F/(R*T);
       double alpha = (1.0f/tanh(Pec)-1.0f/Pec);
       //double pe = (1.0f/tanh(Pec)-1.0f/Pec);
       double c11,c12,c13,c14;
    for (i=1; i <= m_nElements; i++)
    {

        m_ElementData(i).GetNodes(nVNList);

              dMk4.Set(0.0f);
              c11=0;c12=0;c13=0;c14=0;


              c11=0.5*(D(1,i)+D(1,i+1))*m_nZ(1)*(alpha*m_SA0(1,nVNList(1))+(1.0f-
alpha)*m_SA0(1,nVNList(2)));
              c14=0.5*(D(4,i)+D(4,i+1))*m_nZ(4)*(alpha*m_SA0(4,nVNList(1))+(1.0f-
alpha)*m_SA0(4,nVNList(2)));
              c12=0.5*(D(2,i)+D(2,i+1))*m_nZ(2)*((1.0f-
alpha)*m_SA0(2,nVNList(1))+alpha*m_SA0(2,nVNList(2)));
              c13=0.5*(D(3,i)+D(3,i+1))*m_nZ(3)*((1.0f-
alpha)*m_SA0(3,nVNList(1))+alpha*m_SA0(3,nVNList(2)));



              dMk4(1,5) = E*c11/m_fdistance;   dMk4(2,5) = E*c12/m_fdistance;
              dMk4(3,5) = E*c13/m_fdistance;   dMk4(4,5) = E*c14/m_fdistance;
              dMk4(1,10) = -E*c11/m_fdistance; dMk4(2,10) = -E*c12/m_fdistance;
              dMk4(3,10) = -E*c13/m_fdistance; dMk4(4,10) = -E*c14/m_fdistance;
              dMk4(6,5) = -E*c11/m_fdistance;  dMk4(7,5) = -E*c12/m_fdistance;
              dMk4(8,5) = -E*c13/m_fdistance;  dMk4(9,5) = -E*c14/m_fdistance;
              dMk4(6,10) = E*c11/m_fdistance;  dMk4(7,10) = E*c12/m_fdistance;
              dMk4(8,10) = E*c13/m_fdistance;  dMk4(9,10) = E*c14/m_fdistance;
```

```
        // get global degrees-of-freedom associated with element
        int nIndex = 0;
        for (j=1; j <= NUMENODES; j++)
        {
            int n = (nVNList(j)-1)*DOFPN;
            for (k=1; k <= DOFPN; k++)
                    {
                nVEDOF(++nIndex) = n+1;
                            n++;
                     }
        }

        // assemble into structural K
        for (j=1; j <= KSIZE; j++)
        {
            int nRow = nVEDOF(j);
            for (k=1; k <= KSIZE; k++)
            {
                int nCol = nVEDOF(k);
                            m_SSM4(nRow, nCol) += dMk4(j,k);
            }
        }
    }

       // assemble into final K

       MTB.Add(m_SSM1,m_SSM4,Kt);

}

void CMesh::ConstructK_c ()
{
       m_SSM1.Set(0.0f);
    int i, j, k;
    const int KSIZE = NUMENODES*DOFPN;     // size of the stiffness matrix
    CVector<int>    nVEDOF(KSIZE);         // dof associated with element
    CMatrix<double> dMk("k",KSIZE,KSIZE);  // to store the element stiffness
matrix
    CVector<int>    nVNList(NUMENODES);    // list of element nodes
       CMatrix<double> dMk1("k1",KSIZE,KSIZE);
       CMatrix<double> dMk2("k2",KSIZE,KSIZE);

       double A = -F*m_fdistance;
       double E = F/(R*T);


       dMk2.Set(0.0f);



       dMk2(5,1) = A*m_nZ(1)/3.0f;  dMk2(5,2) = A*m_nZ(2)/3.0f;
       dMk2(5,3) = A*m_nZ(3)/3.0f;  dMk2(5,4) = A*m_nZ(4)/3.0f;
       dMk2(5,6) = A*m_nZ(1)/6.0f;  dMk2(5,7) = A*m_nZ(2)/6.0f;
       dMk2(5,8) = A*m_nZ(3)/6.0f;  dMk2(5,9) = A*m_nZ(4)/6.0f;
       dMk2(10,1) = A*m_nZ(1)/6.0f; dMk2(10,2) = A*m_nZ(2)/6.0f;
       dMk2(10,3) = A*m_nZ(3)/6.0f; dMk2(10,4) = A*m_nZ(4)/6.0f;
       dMk2(10,6) = A*m_nZ(1)/3.0f; dMk2(10,7) = A*m_nZ(2)/3.0f;
```

```
   dMk2(10,8) = A*m_nZ(3)/3.0f; dMk2(10,9) = A*m_nZ(4)/3.0f;


// loop thro' all elements
for (i=1; i <= m_nElements; i++)
{

    m_ElementData(i).GetNodes(nVNList);

          dMk1.Set(0.0f);
          dMk.Set(0.0f);



          dMk1(1,1) = 0.5*(D(1,i)+D(1,i+1))/m_fdistance;
          dMk1(2,2) = 0.5*(D(2,i)+D(2,i+1))/m_fdistance;
          dMk1(3,3) = 0.5*(D(3,i)+D(3,i+1))/m_fdistance;
          dMk1(4,4) = 0.5*(D(4,i)+D(4,i+1))/m_fdistance;
          dMk1(5,5) = DC/m_fdistance;
          dMk1(1,6) = -0.5*(D(1,i)+D(1,i+1))/m_fdistance;
          dMk1(2,7) = -0.5*(D(2,i)+D(2,i+1))/m_fdistance;
          dMk1(3,8) = -0.5*(D(3,i)+D(3,i+1))/m_fdistance;
          dMk1(4,9) = -0.5*(D(4,i)+D(4,i+1))/m_fdistance;
          dMk1(5,10) = -DC/m_fdistance;
          dMk1(6,1) = -0.5*(D(1,i)+D(1,i+1))/m_fdistance;
          dMk1(7,2) = -0.5*(D(2,i)+D(2,i+1))/m_fdistance;
          dMk1(8,3) = -0.5*(D(3,i)+D(3,i+1))/m_fdistance;
          dMk1(9,4) = -0.5*(D(4,i)+D(4,i+1))/m_fdistance;
          dMk1(10,5) = -DC/m_fdistance;
          dMk1(6,6) = 0.5*(D(1,i)+D(1,i+1))/m_fdistance;
          dMk1(7,7) = 0.5*(D(2,i)+D(2,i+1))/m_fdistance;
          dMk1(8,8) = 0.5*(D(3,i)+D(3,i+1))/m_fdistance;
          dMk1(9,9) = 0.5*(D(4,i)+D(4,i+1))/m_fdistance;
          dMk1(10,10) = DC/m_fdistance;

          for (int m=1; m<=KSIZE; m++)
          {
                 for (int n=1; n<=KSIZE; n++)
                 {
                        dMk(m,n) = dMk1(m,n) + dMk2(m,n);
                 }
          }


    // get global degrees-of-freedom associated with element
    int nIndex = 0;
    for (j=1; j <= NUMENODES; j++)
    {
        int n = (nVNList(j)-1)*DOFPN;
        for (k=1; k <= DOFPN; k++)
                 {
            nVEDOF(++nIndex) = n+1;
                        n++;
                 }
    }

    // assemble into structural K
    for (j=1; j <= KSIZE; j++)
```

```
        {
            int nRow = nVEDOF(j);
            for (k=1; k <= KSIZE; k++)
            {
                int nCol = nVEDOF(k);
                m_SSM1(nRow, nCol) += dMk(j,k);
            }
        }
    }
}

void CMesh::ConstructC ()
{

       double L = m_dp*m_fdistance/6.0f;

       m_SSC.Set(0.0f);
    int i, j, k;
    const int CSIZE = NUMENODES*DOFPN;     // size of the stiffness matrix
    CVector<int>    nVEDOF(CSIZE);         // dof associated with element
    CMatrix<double> dc("c",CSIZE,CSIZE);   // to store the element stiffness
matrix
    CVector<int>    nVNList(NUMENODES);    // list of element nodes

    // loop thro' all elements
    for (i=1; i <= m_nElements; i++)
    {

            m_ElementData(i).GetNodes(nVNList);

        dc.Set(0.0f);
              dc(1,1) = 2*L; dc(2,2) = 2*L; dc(3,3) = 2*L; dc(4,4) = 2*L;
              dc(1,6) = 1*L; dc(2,7) = 1*L; dc(3,8) = 1*L; dc(4,9) = 1*L;
              dc(6,1) = 1*L; dc(7,2) = 1*L; dc(8,3) = 1*L; dc(9,4) = 1*L;
              dc(6,6) = 2*L; dc(7,7) = 2*L; dc(8,8) = 2*L; dc(9,9) = 2*L;


        // get global degrees-of-freedom associated with element
        int nIndex = 0;
        for (j=1; j <= NUMENODES; j++)
        {
            int n = (nVNList(j)-1)*DOFPN;
            for (k=1; k <= DOFPN; k++)
                    {
                nVEDOF(++nIndex) = n+1;
                            n++;
                    }
        }

        // assemble into structural K
        for (j=1; j <= CSIZE; j++)
        {
            int nRow = nVEDOF(j);
            for (k=1; k <= CSIZE; k++)
            {
                int nCol = nVEDOF(k);
                m_SSC(nRow, nCol) += dc(j,k);
            }
```

```
        }
    }
}

void CMesh::ConstructCcl ()
{
       double Lp = 0;
       double c1,c2;
       c1 = 0; c2 = 0;
       double L = m_dp*m_fdistance/6.0f;
       Lp = (a*m_dden*(1-m_dp)+m_dp)*m_fdistance/6.0f;
       double alpha = (1.0f/tanh(Pec)-1.0f/Pec);
       m_SSC.Set(0.0f);
    int i, j, k;
    const int CSIZE = NUMENODES*DOFPN;     // size of the stiffness matrix
    CVector<int>    nVEDOF(CSIZE);         // dof associated with element
    CMatrix<double> dc("c",CSIZE,CSIZE);   // to store the element stiffness
matrix
    CVector<int>    nVNList(NUMENODES);    // list of element nodes

    // loop thro' all elements
    for (i=1; i <= m_nElements; i++)
    {

          m_ElementData(i).GetNodes(nVNList);

        dc.Set(0.0f);
             dc(1,1) = 2*Lp; dc(2,2) = 2*L; dc(3,3) = 2*L; dc(4,4) = 2*L;
             dc(1,6) = 1*Lp; dc(2,7) = 1*L; dc(3,8) = 1*L; dc(4,9) = 1*L;
             dc(6,1) = 1*Lp; dc(7,2) = 1*L; dc(8,3) = 1*L; dc(9,4) = 1*L;
             dc(6,6) = 2*Lp; dc(7,7) = 2*L; dc(8,8) = 2*L; dc(9,9) = 2*L;


        // get global degrees-of-freedom associated with element
        int nIndex = 0;
        for (j=1; j <= NUMENODES; j++)
        {
            int n = (nVNList(j)-1)*DOFPN;
            for (k=1; k <= DOFPN; k++)
                    {
                nVEDOF(++nIndex) = n+1;
                            n++;
                    }
        }

        // assemble into structural K
        for (j=1; j <= CSIZE; j++)
        {
            int nRow = nVEDOF(j);
            for (k=1; k <= CSIZE; k++)
            {
                int nCol = nVEDOF(k);
                m_SSC(nRow, nCol) += dc(j,k);
            }
        }
    }
}
```

```
void CMesh::ConstructF ()
{
	std::cout<<"Construct F: Done!"<<std::endl;
}

void CMesh::Solve ()
{
	KK.Set(0.0f);
	FF.Set(0.0f);

	CVector<double> A;
	A.SetSize(m_nNodes*DOFPN);
	A.Set(0.0f);

	CVector<double> B;
	B.SetSize(m_nNodes*DOFPN);
	B.Set(0.0f);

	U_new.Set(0.0f);
	U_old.Set(0.0f);


	for (int i=1; i<=m_nNodes*DOFPN; i++)
	{
		U_old(i) = m_U(i);
	}

	ConstructCcl();
	CalculateD();
	ConstructK_c();
	ConstructE();
	ConstructK_t();
	MTB.Scale(Kt,m_fdt);
	MTB.Add(m_SSC,Kt,KK);
	MTB.MatMultVec(m_SSC,U_old,FF);



	// induce boundary condition

	for (int i=1; i<=5; i++)
	{
		FF(i) = m_U(i);
	}

	for (int i=m_nNodes*DOFPN-4; i<=m_nNodes*DOFPN; i++)
	{
		FF(i) = m_U(i);
	}

	for (int i=6; i<=m_nNodes*DOFPN-5; i++)
	{
		for (int j=1; j<=5; j++)
		{
			FF(i) -= KK(i,j)*U_old(j);
		}

		for (int j=m_nNodes*DOFPN-4; j<=m_nNodes*DOFPN; j++)
```

```
        {
                FF(i) -= KK(i,j)*U_old(j);
        }
}

for (int i=1; i<=m_nNodes*DOFPN; i++)
{
        for (int j=1; j<=5; j++)
        {
                if (j==i)
                {
                        KK(i,j) = 1;
                }
                else
                {
                        KK(i,j) = 0;
                        KK(j,i) = 0;
                }
        }

        for (int j=m_nNodes*DOFPN-4; j<=m_nNodes*DOFPN; j++)
        {
                if (j==i)
                {
                        KK(i,j) = 1;
                }
                else
                {
                        KK(i,j) = 0;
                        KK(j,i) = 0;
                }
        }

}


MTB.AxEqb(KK,U_new,FF,TOL);

for (int i=1; i<=m_nNodes*DOFPN; i++)
{
        m_U(i) = U_new(i);
}
for (int i=1; i<=m_nNodes; i++)
{
        m_SA0(1,i) = U_new(5*i-4);
        m_SA0(2,i) = U_new(5*i-3);
        m_SA0(3,i) = U_new(5*i-2);
        m_SA0(4,i) = U_new(5*i-1);
        m_Phi(i)   = U_new(5*i);
}

if (JUDGE == 1)
{
        for (int i=1;i<=m_nNodes;i++)
        {
                double Cl = 0;
                Cl = abs(m_SA0(1,i)/1000*35);
                m_Ccl(i) = a*pow(Cl,b);  // bound chloride, in g/g-solid
```

```
		}
	}
}

void CMesh::Solve2 (int j)
{
	J_old.Set(0.0f);
	for (int i=1;i<=m_nNodes;i++)
	{
		J_old(i) = m_J(j,i);
	}

	m_J(j,1) = -D(j,1)*((-3*m_SA0(j,1)+4*m_SA0(j,2)-
m_SA0(j,3))/(2.0f*m_fdistance)+m_nZ(j)*F*m_dE(1)*m_SA0(j,1)/(R*T));
	for (int i=2;i<m_nNodes;i++)
	{
		m_J(j,i) = -D(j,i)*((m_SA0(j,i+1)-m_SA0(j,i-
1))/(2.0f*m_fdistance)+m_nZ(j)*F*m_dE(i)*m_SA0(j,i)/(R*T));
	}
	m_J(j,m_nNodes) = -D(j,m_nNodes)*((3*m_SA0(j,m_nNodes)-4*m_SA0(j,m_nNodes-
1)+m_SA0(j,m_nNodes-
2))/(2.0f*m_fdistance)+m_nZ(j)*F*m_dE(m_nNodes)*m_SA0(j,m_nNodes)/(R*T));



	for (int i=1;i<=m_nNodes;i++)
	{
		m_Coulombs(j,i) += -
0.25*pow(m_dArea,2)*PI*F*m_nZ(j)*0.5*(m_J(j,i)+J_old(i))*m_fdt;
	}

	if (N%600==0)
	{


		for (int i=1;i<=m_nNodes;i++)
		{
			coulombs(j) += m_Coulombs(j,i);
		}
		coulombs(j) = coulombs(j)/m_nNodes;

		CreateOutputCon(j,m_SA0,N);
		CreateOutputFlux(j,m_J,N);
		CreateOutputCol(j,m_Coulombs,N);
	}
}


void CMesh::Solve3()
{
	if (N%600==0)
	{
		for (int i=1;i<=m_nNodes;i++)
		{

			m_I(i) = -
0.25*pow(m_dArea,2)*PI*F*(m_nZ(1)*m_J(1,i)+m_nZ(2)*m_J(2,i)+m_nZ(3)*m_J(3,i)+m_nZ(
4)*m_J(4,i));
```

```
                }

                coulombs(5) = coulombs(1) + coulombs(2) + coulombs(3) + coulombs(4);
                for (int i=1;i<=4;i++)
                {
                        coulombs(5+i) = coulombs(i)/coulombs(5);
                }
                CreateOutputCol2(7,coulombs,N);
                CreateOutput2(4,m_I,N);
                CreateOutput2(5,m_Phi,N);
                CreateOutput2(6,m_dE,N);
                CreateOutputCon2(8,m_Ccl,N);
        }
}

void CMesh::Counter ()
{
        TIME += m_fdt;
        N +=1;
}

void CMesh::GetStep (int& Step)
{
        Step = N;
}

void CMesh::GetTotal (int& total)
{
        total = m_nStep;
}

void CMesh::Getkinds (int& kind)
{
        kind = m_nKinds;
}


void CMesh::GetTime(float& t)
{
        t = T;
    }
```

APPENDIX B

MATLAB CODE FOR GENERATING 3D CRACK MODELS

```
%%% This program is developed to create 2D images with random cracks
%%% thourgh a concrete cube.
%%% Modified edition: without branching, use crack volume ratio and
%%% tortuosity to control single crack
%%% Author: Pu Yang
%%% Date: November 2013

clc
clear all
close all

%=========================================================================%
%PART A: Basic parameters
%=========================================================================%

%*************************************************************************%
% (1). User Input
%*************************************************************************%

dx = 300;                    % length of 3D image in pixels
dy = 300;                    % width of 3D image in pixels
dz = 300;                    % height of 3D image in pixels

Phi = 0.10                   % total crack volume ratio
W_surface = 30               % effective crack width of surface crack,in
pxl
Tor = 0.3                    % tortuosity of crack
beta   = 0.3                 % height coefficient
N_lay  = 42                  % total zig-zag numbers of perpendicular
crack

tor_dz = 0.5                 % roughness factor in surface
lemda  = 150                 % segment factor for tortuosity calculation


%*************************************************************************%
% (2). Varibles based on input data
%*************************************************************************%

L_total = dz/sqrt(Tor);            % Total effective length
L_eff = L_total - beta*dz;         % effective length starts from notch
W_stable = (dx*dz*Phi - mbeta*dz*0.5*W_surface)/(L_eff+beta*dz*0.5)
% effective crack width in stable level, in pixcel
Num_layer = (1-beta)*dz/N_lay;      % number of layer for each zig_zag
line
slope_surface = zeros(1,60);        % slpoe of zig_zag line
factor_roughness = zeros(1,dz);     % slope of zig_zag line in
perpendicular direction
slope_thru = zeros(1,N_lay);        % slope of zig_zag line in
perpendicular direction, for stable crack
L_zig = zeros(1,N_lay);             % length of each zig_zag line in
perpendicular direction
R = zeros(1,dy);                    % crack radius in each point
Y = zeros(dz,dy);                   % define the crack boundary
R_al = zeros(1,dz-lemda);           % roughness matrix
```

```
w_avg = zeros(1,dz);                % average crack width of each layer
l = 0;                              % initial zig-zag line length
x0 = 0;                             % start point of the surface crack
y0 = 150;                           % start point of the surface crack

%=========================================================================%
%PART B: Random data generation & saving
%=========================================================================%

[fileout, foldout] = uiputfile('bw', 'Pick a folder to write 2D
images' );

Determine_xy = 1;            % choose to use excising data for the
slopes of surface zig_zag line (=0) or create a set of new data (=1)
Determine_dz = 1;            % choose to use excising data for the
slopes of thru zig_zag line (=0) or create a set of new data (=1)
Determine_R  = 1;            % choose to use excising data for the
radius of crack circles (=0) or create a set of new datas (=1)
Determine_z  = 1;            % choose to use excising data for the
slopes of depth zig_zag line (=0) or create a set of new data (=1)

Save_xy = 0;          % choose weither to save new data (=1) or not (=0)
Save_yz = 0;          % choose weither to save new data (=1) or not (=0)
Save_R  = 0;          % choose weither to save new data (=1) or not (=0)
Save_z  = 0;          % choose weither to save new data (=1) or not (=0)

% data generation & saving for all the random generators
if Determine_xy==1
    slope_surface = 0.5*randn(1,60);
% creare random slope for each zig_zag lines, need to choose
appropriate coefficient
    if Save_xy==1
            xlswrite('Random_generater_data',slope_surface',1,'A2:A61');
    else
    end
else
    D = xlsread('Random_generater_data',1,'A2:A61');
    slope_surface = D';
end

if Determine_z==1
    slope_thru = rand(1,N_lay);
% creare basic random numbers for slope for each depth zig_zag lines
    if Save_z==1
    xlswrite('Random_generater_data',slope_thru',1,'C2:C43');
    else
    end
 else
    G = xlsread('Random_generater_data',1,'C2:C43');
    slope_thru = G';
end

if Determine_dz==1
    factor_roughness = randn(1,dz);
% creare basic random numbers for slope for each point on zig_zag lines
```

```
    if Save_yz==1
    xlswrite('Random_generater_data',factor_roughness',1,'B2:B301');
    else
    end
else
    E = xlsread('Random_generater_data',1,'B2:B301');
    factor_roughness = E';
end

if Determine_R==1
    R = rand(1,dx);% create the basic random crack radius in each point
    if Save_R==1
    xlswrite('Random_generater_data',R',2,'A2:A301');
    else
    end
else
    F = xlsread('Random_generater_data',2,'A2:A301');
    R = F';
end


% Modified the random numbers
for i=1:dz
    if abs(factor_roughness(i))<0.5

factor_roughness(i)=factor_roughness(i)+sign(factor_roughness(i))*0.5;
    else if abs(factor_roughness(i))>1
            factor_roughness(i)=factor_roughness(i)-
sign(factor_roughness(i))*0.5;
        end
    end
    factor_roughness(i) = tor_dz*factor_roughness(i);
end

L_sum = 0;
for i=1:N_lay
    slope_thru(i)=slope_thru(i)-0.5;
    L_zig(1,i) = sqrt(slope_thru(i)^2+1)*Num_layer;
    L_sum = L_sum + L_zig(1,i);
end
if L_eff<= L_sum
    slope_thru = zeros(1,N_lay);
else
    tor_ratio = L_eff/L_sum;
    for i=1:N_lay
        L_zig(1,i) = tor_ratio*L_zig(1,i);
        slope_thru(i) =
sign(slope_thru(i))*sqrt((L_zig(1,i)/Num_layer)^2-1);
    end
end


for i=1:dx
    if abs(R(i))<0.5
        R(i)=R(i)+sign(R(i))*0.5;
    else if abs(R(i))>1
```

```
            R(i)=R(i)-sign(R(i))*0.5;
        end
    end
end
w_temp = 0;
for j=1:dx
    w_temp = (2*R(j))^3+w_temp;
end
W_avg = (1/dx*w_temp)^(1/3);
Ratio_surface = W_surface/W_avg;
Ratio_stable = W_stable/W_avg;
alpha  = (Ratio_surface-Ratio_stable)/(beta*dz);    % ratio change rate


%=========================================================================%
%PART C: Crack generation
%=========================================================================%


% start to creat the surface crack on the first layer
for z1 = 1
    A = zeros(dx,dy);

    for x1 = 1:5
        y1 = 150+factor_roughness(1,z1)+slope_surface(1,1)*x1;
        t =Ratio_surface*R(1,x1);
        Y(z1,x1) = y1+t;                    % Record the bc of crack
        for r = 0:t
            for th =0:1:360
                x  = x1+r*cos(th);
                y  = y1+r*(sin(th));
                x=round(x);
                y=round(y);
                if y <= 0
                    y =1;
                end
                if x <= 0
                    x =1;
                end
                A(x,y)= 1;
            end
        end
        l = l + sqrt((x1-x0)^2+(y1-y0)^2);
        x0=x1;
        y0=y1;
    end

    x0=x1;
    y0=y1;

    for n = 1:59
        x0=x1;
        y0=y1;
        b=y0-slope_surface(1,n+1)*x0;
        for x1 = 5*n:1:5*(n+1)
```

```
            y1 = slope_surface(1,n+1)*x1+b; % Record the bc of crack
            t =Ratio_surface*R(1,x1);
            Y(z1,x1) = y1+t;
            for r = 0:t
                for th =0:1:360
                    x  = x1+r*cos(th);
                    y  = y1+r*(sin(th));
                    x=round(x);
                    y=round(y);
                    if y <= 0
                        y =1;
                    end
                    if x <= 0
                        x =1;
                    end
                    A(x,y)= 1;
                end
            end
            l = l + sqrt((x1-x0)^2+(y1-y0)^2);
            x0=x1;
            y0=y1;
        end
    end

    im = double(A(1:300,1:300));
    im1 = mat2gray(im);
    if z1<10
        imwrite(im1,[foldout fileout '00' num2str(z1)
'.tif'],'tif','compression','none')
    else if z1<100
            imwrite(im1,[foldout fileout '0' num2str(z1)
'.tif'],'tif','compression','none')
        else
            imwrite(im1,[foldout fileout num2str(z1)
'.tif'],'tif','compression','none')
        end
    end

    w_1 = 0;
    for j=1:dx
        w_1 = (2*Ratio_surface*R(1,j))^3+w_1;
    end
    w_avg(1,z1)= (1/dx*w_1)^(1/3);

end

% creat crack on the width_change layers

w = Ratio_surface - alpha;

for z1 = 2:1:beta*dz
    A = zeros(dx,dy);

    for x1 = 1:5
        y1 = 150+factor_roughness(1,z1)+slope_surface(1,1)*x1;
```

```
    t =w*R(1,x1);
    Y(z1,x1) = y1+t;        % Record the boundary condition of crack
    for r = 0:t
        for th =0:1:360
            x  = x1+r*cos(th);
            y  = y1+r*(sin(th));
            x=round(x);
            y=round(y);
            if y <= 0
                y =1;
            end
            if x <= 0
                x =1;
            end
            A(x,y)= 1;
        end
    end
    x0=x1;
    y0=y1;
end

x0=x1;
y0=y1;

for n = 1:59
    x0=x1;
    y0=y1;
    b=y0-slope_surface(1,n+1)*x0;
    for x1 = 5*n:1:5*(n+1)
        y1 = slope_surface(1,n+1)*x1+b;
        t =w*R(1,x1);
        Y(z1,x1) = y1+t; % Record the boundary condition of crack
        for r = 0:t
            for th =0:1:360
                x  = x1+r*cos(th);
                y  = y1+r*(sin(th));
                x=round(x);
                y=round(y);
                if y <= 0
                    y =1;
                end
                if x <= 0
                    x =1;
                end
                A(x,y)= 1;
            end
        end
        x0=x1;
        y0=y1;
    end
end

im = double(A(1:300,1:300));
im1 = mat2gray(im);
if z1<10
```

```
        imwrite(im1,[foldout fileout '00' num2str(z1)
'.tif'],'tif','compression','none')
    else if z1<100
            imwrite(im1,[foldout fileout '0' num2str(z1)
'.tif'],'tif','compression','none')
        else
            imwrite(im1,[foldout fileout num2str(z1)
'.tif'],'tif','compression','none')
        end
    end

    w_1 = 0;
    for j=1:dx
        w_1 = (2*w*R(1,j))^3+w_1;
    end
    w_avg(1,z1)= (1/dx*w_1)^(1/3);
    w = w - alpha;
end


% creat cracks on stable layers
N  = 0;
zz = 0;
for z = 1:N_lay
    for z1 = (beta*dz+1+N):1:(beta*dz+N+(1-beta)*dz/N_lay)
        zz = slope_thru(1,z)+zz;
        A = zeros(dx,dy);
        for x1 = 1:5
            y1 = 150+factor_roughness(1,z1)+slope_surface(1,1)*x1+zz;
            t =Ratio_stable*R(1,x1);
            Y(z1,x1) = y1+t;   % Record the boundary condition of crack
            for r = 0:t
                for th =0:1:360
                    x  = x1+r*cos(th);
                    y  = y1+r*(sin(th));
                    x=round(x);
                    y=round(y);
                    if y <= 0
                        y =1;
                    end
                    if x <= 0
                        x =1;
                    end
                    A(x,y)= 1;
                end
            end
            x0=x1;
            y0=y1;
        end

        x0=x1;
        y0=y1;

        for n = 1:59
            x0=x1;
            y0=y1;
```

```
            b=y0-slope_surface(1,n+1)*x0;
            for x1 = 5*n:5*(n+1)
                y1 = slope_surface(1,n+1)*x1+b;
                t =Ratio_stable*R(1,x1);
                Y(z1,x1) = y1+t;% Record the bondary condition of crack
                for r = 0:t
                    for th =0:1:360
                        x  = x1+r*cos(th);
                        y  = y1+r*(sin(th));
                        x=round(x);
                        y=round(y);
                        if y <= 0
                            y =1;
                        end
                        if x <= 0
                            x =1;
                        end
                        A(x,y)= 1;
                    end
                end
                x0=x1;
                y0=y1;
            end
        end

        im = double(A(1:300,1:300));
        im1 = mat2gray(im);
        if z1<10
            imwrite(im1,[foldout fileout '00' num2str(z1)
'.tif'],'tif','compression','none')
        else if z1<100
                 imwrite(im1,[foldout fileout '0' num2str(z1)
'.tif'],'tif','compression','none')
            else
                 imwrite(im1,[foldout fileout num2str(z1)
'.tif'],'tif','compression','none')
            end
        end


        w_1 = 0;
        for j=1:dx
            w_1 = (2*Ratio_stable*R(1,j))^3+w_1;
        end
        w_avg(1,z1)= (1/dx*w_1)^(1/3);

    end
    N = N+(1-beta)*dz/N_lay;
end

%=========================================================================%
%PART D: Parameters calculation
%=========================================================================%

%calculate permeability
```

```
l_eff = l                                          % effective length of
surface crack


l_thru = L_total                                   % effective length of
prependicular crack

C = 0;
for z1 = 1:dz
    C = C+(1/(w_avg(1,z1))^3);
end
w_eff = (dz/C)^(1/3);                               % effective width of
the crack

% calculate global roughness

z = zeros(1,300);
for i=1:dz
    z(1,i)=i;
end

for i=1:(dz-lemda)
    m = (Y(i+lemda,1)-Y(i,1))/lemda;
    alpha = atan(m);
    p = 0;
    for j=i:(i+lemda)
        p=p+abs(m*z(1,j)+Y(i,1)-m*z(1,i)-Y(j,1));
    end
    R_al(1,i)=1/lemda*p*cos(alpha);
end

R_ag = 1/(dz-lemda)*sum(R_al);      % global surface roughness

R_r = R_ag/(2*w_eff)                % relative surface roughness

taun = (dz/l_thru)^2                % turtuosity factor

R_ag = R_ag*5                       % change the unit from pixel to um

w_eff = w_eff*5                     % change the unit from pixel to um

K = taun*(w_eff)^2/(12*(1+8.8*R_r^1.5))     % permeability

Width_s = w_avg(1,1)*5;

Width_b = w_avg(1,dz)*5;

V = dy*((w_avg(1)+w_avg(dz))/2*beta*dz+w_avg(dz)*(1-
beta)*dz)/(dx*dy*dz);

%=========================================================================%
%PART E: Saving results
%=========================================================================%
```

```
xlswrite('Permeability_data',K,1,'A1');
xlswrite('Permeability_data',w_eff,1,'B1');
xlswrite('Permeability_data',taun,1,'C1');
xlswrite('Permeability_data',R_ag,1,'D1');
xlswrite('Permeability_data',Width_s,1,'E1');
xlswrite('Permeability_data',Width_b,1,'F1');
xlswrite('Permeability_data',V,1,'G1');       % record the test data


% plot the crack width - crack depth curve
plot(1:dz,w_avg)
title('Crack width - Crack depth relationship')
xlabel('Crack depth (pixels)')
ylabel('Crack width (pixels)')
axis([0 320 0 40]); % xmin, xmax, ymin, ymax
```

APPENDIX C

MATLAB CODE FOR D2Q9-BGK LATTICE BOLTZMANN METHOD

```
% Gianni Schena  July 2005, schena@units.it
% Lattice Boltzmann LBE, geometry: D2Q9, model: BGK
% Application to permeability in porous media

Restart=false % to restart from an earlier convergence
logical(Restart);

if Restart==false;
close all, clear all % start from scratch and clean ...
Restart=false;
% type of channel geometry ;
% one of the flollowing flags == true
Pois_test=true, % no obstacles in the 2D channel
% porous systems
obs_regolare=false %
obs_irregolare=false %
tic
%   IN
% |vvvv|    + y
% |vvvv|     ^
% |vvvv|     | -> + x
%  OUT

% Pores in 2D : Wet and Dry locations (Wet ==1 , Dry ==0 )
wXh_Dry=[3,1];wXh_Wet=[3,4];

if obs_regolare, % with internal obstacles

A=repmat([zeros(wXh_Dry),ones(wXh_Wet)],[1,3]);A=[A,zeros(wXh_Dry)];
B=ones(size(A));
C=[A;B]  ; D=repmat(C,4,1);
D=[B;D]
end

if obs_irregolare, % with int obstacles
A1=repmat([zeros(wXh_Dry),ones(wXh_Wet)],[1,3]);
A1=[A1,zeros(wXh_Dry)]  ;
B=ones(size(A1));
C1=repmat([ones(wXh_Wet),zeros(wXh_Dry)],[1,3]); C1=[C1,ones(wXh_Dry)];
E=[A1;B;C1;B];
D=repmat(E,2,1);
D=[B;D]
end

if ~Pois_test
figure,imshow(D,[])
Channel2D=D;
Len_Channel_2D=size(Channel2D,1); % Length
Width=size(Channel2D,2); % should not be hod
Channel_2D_half_Width=Width/2,
end

% test without obstacles (i.e. 2D channel & no obstacles)

if Pois_test
```

```
%over-writes the definition of the pore space
clear Channel2D
Len_Channel_2D=36, % lunghezza canale 2d
Channel_2D_half_Width=8; Width=Channel_2D_half_Width*2;
Channel2D=ones(Len_Channel_2D,Width); % define wet area
%Channel2D(6:12,6:8)=0; % put fluid obstacle
imshow(Channel2D,[]);
end

[Nr Mc]=size(Channel2D); % Number rows and Munber columns

% porosity
porosity=nnz(Channel2D==1)/(Nr*Mc)



% FLUID PROPERTIES
% physical properties
cs2=1/3; %
cP_visco=1; % [cP] 1 CP Dinamic water viscosity 20 C
density=1.; % fluid density
Lky_visco=cP_visco/density; % lattice kinematic viscosity
omega=(Lky_visco/cs2+0.5).^-1; %  omega: relaxation frequency
%Lky_visco=cs2*(1/omega - 0.5) , % lattice kinematic viscosity
%dPdL= Pressure / dL;% External pressure gradient [atm/cm]

uy_fin_max=-0.2;
%dPdL = abs( 2*Lky_visco*uy_fin_max/(Channel_2D_half_Width.^2) );
dPdL=-0.0125;
uy_fin_max=dPdL*(Channel_2D_half_Width.^2)/(2*Lky_visco); % Poiseuille
Gradient;
% max poiseuille final  velocity on the flow profile
uy0=-0.001; ux0=0.0001; %  linear vel .. inizialization

%
% uy_fin_max=-0.2; % max poiseuille final  velocity on the flow profile
% omega=0.5, cs2=1/3; % omega: relaxation frequency
% Lky_visco=cs2*(1/omega - 0.5) , % lattice kinematic viscosity
% dPdL = abs( 2*Lky_visco*uy_fin_max/(Channel_2D_half_Width.^2) ); %
Poiseuille Gradient;
%

uyf_av=uy_fin_max*(2/3);; % average fluid velocity on the profile

x_profile=([-Channel_2D_half_Width:+Channel_2D_half_Width-1]+0.5);
uy_analy_profile=uy_fin_max.*(1-  ( x_profile
/Channel_2D_half_Width).^2 ); % analytical velocity profile

av_vel_t=1.e+10; % inizialization (t=0)
%PixelSize= 5; % [Microns]
%dL=(Nr*PixelSize*1.0E-4); % sample hight [cm]



%
% EXPERIMENTAL SET-UP
% inlet and outlet buffers
```

```
inb=2, oub=2; % inlet and outlet buffers thickness
% add fluid at the inlet (top) and outlet (down)
inlet=ones(inb,Mc); outlet=ones(oub,Mc);
Channel2D=[ [inlet]; Channel2D ;[outlet] ] ; % add flux in and down (E
to W)
[Nr Mc]=size(Channel2D); % update size
% boundaries related to the experimental set up
wb=2; % wall thickness
Channel2D=[zeros(Nr,wb), Channel2D , zeros(Nr,wb)]; % add walls (no
fluid leak)
[Nr Mc]=size(Channel2D); % update size
uy_analy_profile=[zeros(1,wb), uy_analy_profile, zeros(1,wb) ] ; % take
into account walls
x_pro_fig=[[x_profile(1)-[wb:-1:1]], [x_profile,
[1:wb]+x_profile(end)] ];

% Figure plots analytical parabolic profile
figure(20), plot(x_pro_fig,uy_analy_profile,'-'), grid on,
title('Analytical parab. profile for Poiseuille planar flow in a
channel')


% VISUALIZE PORE SPACE & FLUID OSTACLES & MEDIAL AXIS
figure, imshow(Channel2D); title('Vassel geometry');
Channel2D=logical(Channel2D);
% obstacles for Bounce Back ( in front of the grain)
Obstacles=bwperim(Channel2D,8); % perimeter of the grains for bounce
back Bound.Cond.
border=logical(ones(Nr,Mc));
border([1:inb,Nr-oub:Nr],[wb+2:Mc-wb-1])=0;
Obstacles=Obstacles.*(border);
figure, imshow(Obstacles); title(' Fluid obstacles (in the fluid)' );
%
Medial_axis=bwmorph(Channel2D,'thin',Inf); %
figure, imshow(Medial_axis); title('Medial axis');
figure(10) % used to visualize evolution of rho
figure(11) % used to visualize ux
figure(12) % used to visualize uy (i.e. top -> down)

% INDICES
% Wet locations etc.
[iabw1 jabw1]=find(Channel2D==1); % indices i,j, of active lattice
locations i.e. pore
lena=length(iabw1); % number of active location i.e. of pore space
lattice cells
ija= (jabw1-1)*Nr+iabw1; % equivalent single index (i,j)->> ija for
active locations
% absolute (single index) position of the obstacles in for bounce back
in Channel2D
% Obstacles
[iobs jobs]=find(Obstacles);lenobs=length(iobs); ijobs= (jobs-
1)*Nr+iobs; % as above
% Medial axis of the pore space
[ima jma]=find(Medial_axis); lenma=length(ima);  ijma= (jma-1)*Nr+ima; %
as above
% Internal wet locations : wet & ~obstables
```

```
% (i.e. internal wet lattice location non in contact with dray
locations)
[iawint jawint]=find(( Channel2D==1 & ~Obstacles)); % indices i,j, of
active lattice locations
lenwint=length(iawint); % number of internal (i.e. not border) wet
locations
ijaint= (jawint-1)*Nr+iawint; % equivalent singl
NxM=Nr*Mc;

% DIRECTIONS: E N W S NE NW SW SE ZERO (ZERO:Rest Particle)
%    y^
%  6 2 5           ^         NW  N  NE
%  3 9 1 ... +x-> +y         W   RP  E
%  7 4 8                     SW  S  SE
%   -y
% x & y components of velocities , +x is to est , +y is to nord
East=1; North=2; West=3; South=4; NE=5; NW=6; SW=7; SE=8; RP=9;
N_c=9 ; % number of directions
% versors D2Q9
C_x=[1 0 -1  0 1 -1 -1  1 0];
C_y=[0 1  0 -1 1  1 -1 -1 0]; C=[C_x;C_y]

% BOUNCE BACK SCHEME
% after collision the fluid elements densities f are sent back to the
% lattice node they come from with opposite direction
% indices opposite to 1:8 for fast inversion after bounce
ic_op = [3 4 1 2 7 8 5 6]; %   i.e. 4 is opposite to 2 etc.

% PERIODIC BOUNDARY CONDITIONS - reinjection rules
yi2=[Nr , 1:Nr , 1]; % this definition allows implemening Period Bound
Cond
%yi2=[1, Nr , 2:Nr-1 , 1,Nr]; % re-inj the second last to as first
% directional weights (density weights)
w0=16/36. ; w1=4/36. ; w2=1/36.;
W=[ w1 w1 w1 w1 w2 w2 w2 w2 w0];
%c constants (sound speed related)
cs2=1/3; cs2x2=2*cs2; cs4x2=2*cs2.^2;
f1=1/cs2; f2=1/cs2x2; f3=1/cs4x2;
f1=3., f2=4.5; f3=1.5; % coef. of the f equil.

% declarative statemets
f=zeros(Nr,Mc,N_c); % array of fluid density distribution
feq=zeros(Nr,Mc,N_c); % f at equilibrium
rho=ones(Nr,Mc); % macro-scopic density
temp1=zeros(Nr,Mc);
ux=zeros(Nr,Mc);   uy=zeros(Nr,Mc); uyout=zeros(Nr,Mc);  %
dimensionless velocities
uxsq=zeros(Nr,Mc); uysq=zeros(Nr,Mc);   usq=zeros(Nr,Mc);  % higher
degree velocities

% initialization arrays : start values in the wet area
for ia=1:lena % stat values in the active cells only ; 0 outside
    i=iabw1(ia);  j=jabw1(ia);
    f(i,j,:)=1/9; % uniform density distribution for a start
end
uy(ija)=uy0; ux(ija)=ux0; % initialize fluid velocities
```

```
rho(ija)=density;

% EXTERNAL (Body) FORCES e.g. inlet pressure or inlet-outlet gradient
% directions: E N W S NE NW SW SE ZERO
force = -dPdL*(1/6)*1*[0 -1 0 1 -1 -1 1  1  0]'; %;
%...                    E  N E S NE NW SW SE RP ...
% the pressure pushes the fluid down i.e. N to S

% While .. MAIN TIME EVOLUTION LOOP
StopFlag=false; % i.e. logical(0)
Max_Iter=3000; % max allowed number of iteration
Check_Iter=1; Output_Every=20; % frequency of check & output
Cur_Iter=0; % current iteration counter inizialization
toler=1.0e-8; % tollerance to declare convegence
Cond_path=[]; % recording values of the convergence criterium
density_path=[]; % recording aver. density values for convergence
end % ends if restart

if(Restart==true)
 StopFlag=false;  Max_Iter=Max_Iter+3000; toler=1.0e-12;
end



while(~StopFlag)
    Cur_Iter=Cur_Iter+1 % iteration counter update

    % density and moments
    rho=sum(f,3); % density

    if Cur_Iter >1 % use inizialization ux uy to start
        % Moments ... Note:C_x(9)=C_y(9)=0
        ux=zeros(Nr,Mc); uy=zeros(Nr,Mc);
        for ic=1:N_c-1;
            ux = ux + C_x(ic).*f(:,:,ic) ; uy = uy +
C_y(ic).*f(:,:,ic)  ;
        end
       % uy=f(:,:,2) +f(:,:,5)+f(:,:,6)-f(:,:,4)-f(:,:,7)-f(:,:,8); %
in short !
       % ux=f(:,:,1) +f(:,:,5)+f(:,:,8)-f(:,:,3)-f(:,:,6)-f(:,:,7); %
in short !
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ux(ija)=ux(ija)./rho(ija); uy(ija)=uy(ija)./rho(ija);
    uxsq(ija)=ux(ija).^2; uysq(ija)=uy(ija).^2;
    usq(ija)=uxsq(ija)+uysq(ija); %

    % weighted densities : rest particle, principal axis, diagonals
    rt0 = w0.*rho; rt1 = w1.*rho; rt2 = w2.*rho;

    % Equilibrium distribution
    % main  directions ( + cross)
    feq(ija)= rt1(ija) .*(1 +f1*ux(ija) +f2*uxsq(ija) -f3*usq(ija));
    feq(ija+NxM*(2-1))= rt1(ija) .*(1 +f1*uy(ija) +f2*uysq(ija) -
f3*usq(ija));
```

```
    feq(ija+NxM*(3-1))= rt1(ija) .*(1 -f1*ux(ija) +f2*uxsq(ija) -
f3*usq(ija));
    %feq(ija+NxM*(3)=f(ija)-2*rt1(ija)*f1.*ux(ija); % much faster... !!
    feq(ija+NxM*(4-1))= rt1(ija) .*(1 -f1*uy(ija) +f2*uysq(ija) -
f3*usq(ija));

    % diagonals (X diagonals) (ic-1)
    feq(ija+NxM*(5-1))= rt2(ija) .*(1 +f1*(+ux(ija)+uy(ija))
+f2*(+ux(ija)+uy(ija)).^2 -f3.*usq(ija));
    feq(ija+NxM*(6-1))= rt2(ija) .*(1 +f1*(-ux(ija)+uy(ija)) +f2*(-
ux(ija)+uy(ija)).^2 -f3.*usq(ija));
    feq(ija+NxM*(7-1))= rt2(ija) .*(1 +f1*(-ux(ija)-uy(ija)) +f2*(-
ux(ija)-uy(ija)).^2 -f3.*usq(ija));
    feq(ija+NxM*(8-1))= rt2(ija) .*(1 +f1*(+ux(ija)-uy(ija))
+f2*(+ux(ija)-uy(ija)).^2 -f3.*usq(ija));
    % rest particle (.) ic=9
    feq(ija+NxM*(9-1))= rt0(ija) .*(1 - f3*usq(ija));

    %Collision (between fluid elements)omega=relaxation frequency
    f=(1.-omega).*f + omega.*feq;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %add external body force due to the pressure gradient prop. to dPdL
    for ic=1:N_c;%-1
        for ia=1:lena
            i=iabw1(ia);  j=jabw1(ia);
            % if Obstacles(i,j)==0 % the i,j is not aderent to the
boundaries
            % if ( f(i,j,ic) + force(ic) ) >0; %! avoid negative
distributions
            %i=1 ;% force only on the first row !
            f(i,j,ic)= f(i,j,ic) + force(ic);
            % end
            % end
        end
    end



    % % STREAM
    % Forward Propagation step & % Bounce Back (collision fluid with
obstacles)
    %f(:,:,9) = f(:,:,9); % Rest element do not move

    feq = f; % temp storage of f in feq
        for ic=1:1:N_c-1, % select velocity layer

        ic2=ic_op(ic); % selects the layer of the velocity opposite to
ic for BB
        temp1=feq(:,:,ic); %

        % from wet location that are NOT on the border to other wet
locations
        for ia=1:1:lenwint % number of internal (i.e. not border) wet
locations
```

```
            i=iawint(ia);  j=jawint(ia);  % so that we care for the wet
space only !
            i2 = i+C_y(ic); j2 = j+C_x(ic); % Expected final locations
to move
            i2=yi2(i2+1); % i2 corrected for PBC when necessary (flow
out re-fed to inlet)
            % i.e the new position (i2,j2) is sure another wet location
            % therefore normal propagation from (i,j) to (i2,j2) on
layer ic
            f(i2,j2,ic)=temp1(i,j); % see circshift(..) fnct for
circularly shifts
        end ; % i and j single loop


        % from wet locations that ARE on the border of obstacles
        for ia=1:1:lenobs % wet border locations
            i=iobs(ia);  j=jobs(ia);  % so that we care for the wet
space only !
            i2 = i+C_y(ic); j2 = j+C_x(ic); % Expected final locations
to move
            i2=yi2(i2+1); % i2 corrected for PBC

            if( Channel2D(i2,j2) ==0 ) % i.e the new position (i2,j2)
is dry
                f(i,j,ic2) =temp1(i,j); % invert direction: bounce-back
in the opposite direction ic2
            else % otherwise, normal propagation from (i,j) to (i2,j2)
on layer ic
                f(i2,j2,ic)=temp1(i,j); % see circshift(..) fnct for
circularly shifts
            end ; % b.b. and propagations

        end ; % i and j single loop
        % special treatment for Corners
        %   f(1,wb+1,ic)=temp1(Nr,Mc-wb);      f(1,Mc-
wb,ic)=temp1(Nr,wb+1);
        %   f(Nr,wb+1,ic)=temp1(1,Mc-wb);      f(Nr,Mc-
wb,ic)=temp1(1,wb+1);

    end ; %  for ic direction

    % ends of Forward Propagation step &  Bounce Back Sections

    % re-calculate  uy as uyout for convergence
    rho=sum(f,3); % density
    % check velocity
    uyout= zeros(Nr,Mc);
    for ic=1:N_c-1;
        uyout= uyout + C_y(ic).*f(:,:,ic) ; % flow dim.less velocity
out
    end
   % uyout(ija)=uyout(ija)./rho(ija); % from momentum to velocity

    % Convergence check on velocity values
```

```
    if (mod(Cur_Iter,Check_Iter)==0) ; % check for convergence every
'Check_Iter' iterations

        % variables monitored
        % mean density and
        vect=rho(ija); vect=vect(:);
        cur_density=mean(vect);
        % mean 'interstitial' velocity
        % uy(ija)=uy(ija)/rho(ija); ?
        vect=uy(ija); av_vel_int= mean(vect)  ; % seepage velocity (in
the wet area)
        % on the whole cross-sectional area of flow (wet + dry)
        av_vel_int=av_vel_int*porosity, % av. vel. on the wet + dry
area
        %av_vel_int=mean2(uy),
        av_vel_tp1 = av_vel_int;
        Condition=abs( abs(av_vel_t/av_vel_tp1 )-1), % should --> 0

        Cond_path=[Cond_path, Condition]; % records the convergence
path (value)
        density_path=[density_path, cur_density];
        %
        av_vel_t=av_vel_tp1; % time t & t+1

        if (Condition < toler) | (Cur_Iter > Max_Iter)
            StopFlag=true;
            display( 'Stop iteration: Convergence met or iteration
exeeding the max allowed' )
            display( ['Current iteration: ',num2str(Cur_Iter),...
                ' Max Number of iter: ',num2str(Max_Iter)] )
            break % Terminate execution of WHILE .. exit the time
evolution loop.

        end    % if(Condition < toler

    end

    if (mod(Cur_Iter,Output_Every)==0) ;  % Output from loop every ...
        %if (Cur_Iter>60) ;  % Output from loop every ...

        rho=sum(f,3); % density
        figure(10); imshow(rho,[0.1 0.9]); title(' rho'); % visualize
density evolution
        figure(11); imshow(ux,[ ]); title(' ux' ); % visualize fluid
velocity horizontal
        figure(12); imshow(-uy,[ ]); title(' uy' ); % visualize fluid
velocity down
        figure(14), imshow(-uyout,[]), title('uyout'); % vis vel flow
out
        up=2; % linear section to visualize up from the lower row
        figure(15), hold off, feather(ux(Nr-up,:),uy(Nr-up,:)),
        figure(15), hold on , plot(uy_analy_profile,'r-')
        title('Analytical vs LB calculated, fluid velocity parabolic
profile')
        pause(3); % time given to visualize properly
```

```
    end % every


   % pause(1);


end %  End main time Evolution Loop

% Output & Draw after the end of the time evolution

figure, plot(Cond_path(2:end)); title('convergence path')
%figure, plot(density_path(2:end)); title('density convergence path')
figure, plot( [uy(Nr-up,:)-uy_analy_profile] ); title('difference : LB
- Analytical solution')

toc

% Permeability K

K_Darcy_Porous_Sys= (av_vel_int*porosity)/dPdL*Lky_visco ,

K_Analy_2D_Channel=(Width^2)/12
```