Mathematical Modelling of Agglomerate Scale Phenomena in Heap Bioleaching

by

Nneoma Ogbonna

Thesis presented for the degree of
Masters in Applied Science
in the Department of Chemical Engineering
University of Cape Town

March 2006

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own.

Signed

Signed.....

Date 27 March 2006

Nneoma Ogbonna

Abstract

Bioleaching is a naturally occurring process that has been harnessed in metal recovery from low grade ores. The heap bioleaching technique involves complex interactions between chemical reactions, microbial processes and transport processes. The need for efficient heap operations has led to the scientific investigation of heap bioleaching, and the development of mathematical models for the process. Over time, the focus of heap leach modelling has moved from models that emphasize particle scale processes to models that emphasize bulk scale processes. In many cases however, the particle scale effects in these bulk scale models are quite simplified.

This thesis aims to provide a means for the systematic integration of particle (or micro-) scale processes into bulk (or macro-) scale models for heap bioleaching, by the development of an intermediate (or meso-) scale "agglomerate" model. The agglomerate is defined as a unit volume of a heap that comprises a solid phase (a size distribution of ore particles), a liquid phase (stagnant and flowing leaching solution, which contains dissolved solutes, attached and planktonic microbes) and a gas phase (flowing air and air pockets). The processes incorporated into the proposed model include reagent diffusion and reaction in a size distribution of ore particles, microbial attachment, detachment and oxidation processes, and the transport of chemical and microbial species to and from the agglomerate. Isothermal agglomerate conditions, and a uniform distribution of reagents in the stagnant liquid phase, are among the modelling assumptions made.

The agglomerate model is applied to investigate the meso-scale bioleaching of a theoretical case study ore that contains mainly chalcocite and pyrite, in the presence of iron oxidizing microbes. The numerical implementation of the model is done in the Python programming language. The integrity of the numerical results is confirmed by performing mass balance checks at the end of each simulation.

Simulation results suggest that as chalcocite leaching progresses deeper into the coarse ore particles, its high intrinsic conversion rate is retarded due to the presence of unreacted covellite and pyrite in outer regions of the particle. The sensitivity of the model with respect to copper recovery relative to the base case results was investigated for the following parameters: temperature, particle size distribution, solution flow rate, inlet substrate concentration, mass transfer parameters, Arrhenius rate constant, inlet iron oxi-

dizing microbial population, maximum iron oxidizing microbial growth rate, and presence of sulfur oxidizers. The sensitivity studies show that the following parameters have the most effect on increasing copper recovery above the base case value: flow rate, inlet ferric concentration, inlet oxygen concentration, and the proportion of fine ore particles.

The study of model sensitivity to temperature suggests the possibility of higher copper recovery by operating in a temperature range below the optimum temperature of the iron oxidizing microbes.

Acknowledgements

I would like to express my gratitude to my supervisors, Dr. Jochen Petersen and Dr. Henri Laurie, for helping me appreciate this project from both an engineering and a mathematical perspective. Thank you for the many brain-storming sessions, and your guidance and support throughout this project.

Many thanks to my family and friends, whose support and encouragement made this experience all the more worthwhile.

I would also like to acknowledge the financial support provided by the African Institute for Mathematical Sciences South Africa, AMIRA International, and the Department of Chemical Engineering UCT, towards my Masters studies.

Table of Contents

List of Figures

List of Tables

List of Codes

Nomenclature

Greek Letters

ϵ	Particle porosity	
μ	Microbial growth rate	1/min
ν	Stoichiometry coefficient	
ρ	Ore density	g/cm^3
ε_b	Stagnant liquid volume fraction	
ε_f	Flowing liquid volume fraction	
ε_p	Particle volume fraction	

Roman Letters

C	Concentration	mol/cm^3
D	Diffusion coefficient	cm^2/min
E	Activation energy	J/mol
g	Mineral grade	mol/g
k_A	Ferric mass transfer parameter	mol/L
k_B	Ferric reduction parameter	mol/L
K_i	Monod constant for substrate i	mol/L or mol/g
K_a	Adsorption equilibrium constant	L/mol
k^*_bf	Mass transfer rate (bulk/flowing interface)	1/min
k_bs	Mass transfer coefficient (particle surface/bulk interface)	cm/min
k_death	Microbial specific death rate	1/min
K_I	Cell inhibition factor	mol/L
k_m	Microbial maintenance rate	mol substrate used/cell/min
k_ref	Arrhenius constant	1/min
N	Microbial population	cells/cm^3
R	Gas constant, see equation (3.16)	J/mol/K
r	Particle radius	cm
S	Source term	mol/cm^3/min
T	Temperature	K
t	Time	min
u	Superficial velocity	cm/min

w	Mass fraction	
X	Mineral conversion	
Y	Microbial yield coefficient	cells produced/mol substrate oxidized

Subscripts

a	Adsorbed
b	Bulk/stagnant liquid
d	Desorbed
f	Flowing liquid
p	Particle

CHAPTER 1

Introduction

1.1 Background

Bioleaching refers to the conversion of an insoluble metal (usually a metal sulfide) into soluble form (usually the metal sulphate), accompanied by biological oxidation and complexation processes (Rawlings, 2002; Rohwerder et al., 2003). Copper is the primary metal targeted by this technique, although it is also considered for zinc, cobalt and nickel recovery (Brierley and Brierley, 2001). In bioleaching, the metal being recovered is solubilized. A similar process to bioleaching is biooxidation. Biooxidation refers to the process in which the recovery of a metal is enhanced by microbial decomposition of the mineral, but the metal being recovered is not solubilized (Rawlings, 2002). An example is the recovery of gold and silver from refractory sulfidic ores, where the activity of leaching bacteria is applied only to remove interfering metal sulfides from ores bearing the precious metals prior to cyanidation treatment. Biomining is a general term that may be used to refer to both processes (Rawlings, 2002).

Bioleaching offers several advantages over conventional physico-chemical methods for ore processing, some of which are listed below (Brierley and Brierley, 2001; Kinnunen, 2004; Rawlings, 2002):

- It does not produce sulfur dioxide, or other harmful emmisions
- There is no need for large amounts of energy like in roasting or smelting.
- It offers low capital and operating costs, especially in metal recovery from low-grade ores, where the metals are not economically recoverable by non-biological methods.
- Reagents (ferric ions and sulfuric acid) are regenerated in the heap under the action of naturally occurring bacteria.
- It offers low technological skills requirement, operational simplicity, and shorter construction times.

The major disadvantages of bioleaching are (Kinnunen, 2004):

- Long reaction times.
- Dependence on climatic conditions (in heap bioleaching).
- Heavy metal toxicity to micro-organisms.
- The potential for acid and metal into leak to the environment .

Since 1980 at least 13 copper bioleach operations have been commissioned, and at least seven plants have been commissioned for the biooxidation pre-treatment of sulfidic-refractory gold concentrates (Brierley and Brierley, 2001; Olson et al., 2003). In addition, several technologies have been developed for the biooxidation of gold, such as BacTech (by BachTech, Australia), Genmin (by GENCOR, South Africa), BIOX (by BMS, Inc.), and GEOCOAT (by Geobiotics, Colorado; also used for copper, zinc and nickel).

1.2 Bioleaching Techniques

The main bioleaching techniques are *in situ*, dump, heap, vat and tank leaching (Kinnunen, 2004). *In situ*, dump and heap bioleaching are irrigation type processes applied mostly to copper recovery, whereas vat and tank bioleaching take place in some form of vessel. Gold biooxidation pre-treatment is primarily by means of tank bioleaching, but it has been demonstrated that pre-treatment of lower value, refractory, whole gold ores can be conducted in heaps (Brierley, 1997). The key characteristics of these leaching techniques are summarized in Fig.1.1.

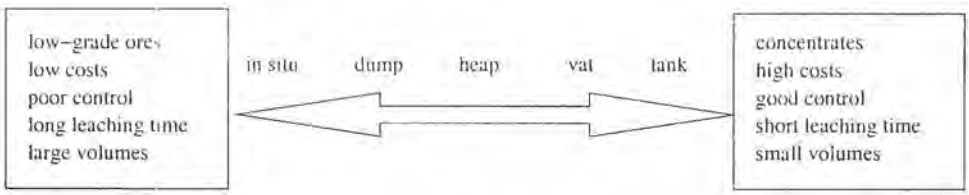


Figure 1.1: Characteristics of different bioleaching techniques (Kinnunen, 2004).

***In situ* bioleaching** refers to the process where ore is treated in-place without mining from the host rock. The ore is fractured by blasting or natural processes, thus producing voids and porosity to allow free solution flow. The pregnant leach solution is collected generally at the bottom of the mine and processed for metal recovery. The application of this process has not been widespread, as it requires very specific ore body characteristics (high permeability ore with low permeability host rock).

Dump bioleaching is used for untreated, uncrushed, run-of-mine rock piles which were previously considered waste. The ore is stacked, and acidified leaching solution applied to the top percolates through the dump, reacting with the mineral. The leaching solution, rich in dissolved metal, is collected at the bottom of the dump and processed for metal recovery. The resulting raffinate is recycled for further dump irrigation.

Heap bioleaching is similar to dump leaching, but it is designed to be more efficient and better controlled. The ore is crushed, agglomerated with sulphuric acid to prevent segregation of particles, and placed on prepared impermeable leaching pads before irrigation with leaching solution (Rawlings, 2002). The process involves forced aeration of the heap, to enhance microbial oxidizing ability. The leaching solution may be inoculated with bacteria. However, bioleaching bacteria are ubiquitous, so it is not clear whether inoculation speeds up the process (Rawlings, 2002).

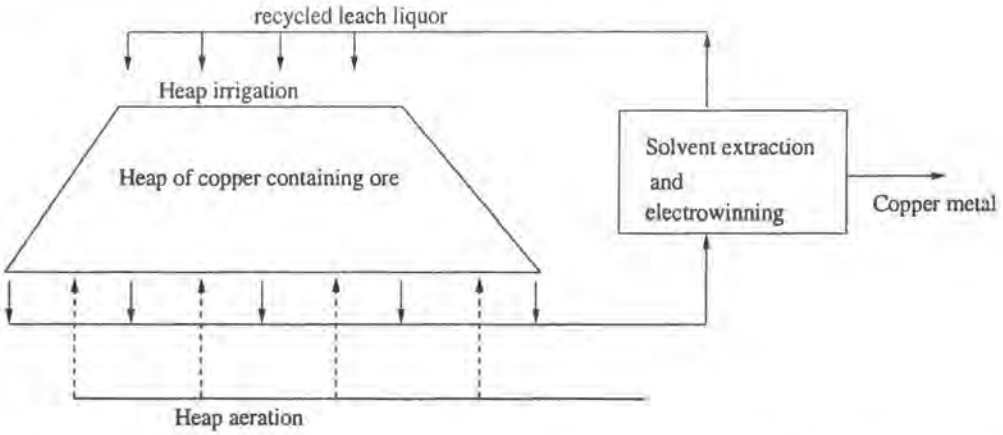


Figure 1.2: Heap leaching of copper-containing ore (adapted from (Rawlings, 2002)).

Tank bioleaching is used for high-value ores and concentrates, since it has the highest construction and operating costs (Kinnunen, 2004; Rawlings, 2002). Here, leaching of fine ore particles takes place in aerated, agitated, stirred tank bioreactors. The tanks are typically arranged in series and are operated in continuous-flow mode, with feed being added to the first tank and overflowing from tank to tank, until the leaching of the mineral concentrate is sufficiently complete (Rawlings, 2002).

Vat bioleaching is a hybrid of heap and tank bioleaching - crushed ore is leached in concrete vats with acid-proof material. This technique does not involve agitation; rather the vats (containing crushed ore) are flooded with the leaching solution. The vats provide for better control of the leaching environment while not having the expensive requirement

of agitation and air distribution.

This research is concerned with the mathematical modelling of heap bioleaching. A brief overview of heap bioleach modelling, and the motivation for this study, are given in the section below.

1.3 Motivation

Heap bioleaching of marginal sulfide ores, especially chalcocite, has become increasingly popular with the decline of available high-grade deposits (Petersen and Dixon, 2003). However, many of the parameters in heap bioleaching, such as temperature, pH, and availability of oxygen and carbon dioxide, are not well controlled (Petersen and Dixon, 2003).

Scientific investigation of heap bioleaching has led to the development of mathematical models for the process, some of which are reviewed and compared by Dixon (2003). Ultimately, the purpose of a heap model is to improve the design and optimization of heap leaching operations, and to act as a diagnostic tool for existing heaps. Most of the early heap leaching models dealt with leaching at the particle scale (Bartlett, 1992a; Braun et al., 1974; Davis et al., 1986; Davis and Ritchie, 1986, 1987; Roman and Olsen, 1974; Shafer et al., 1979). On the other hand, more recent bioleaching models emphasize the effects of bulk scale phenomena, such as liquid flow, gas flow, and temperature distribution, on heap performance (Dixon, 2000; Dixon and Petersen, 2003; Leahy et al., 2003, 2005a,b; Moreno et al., 1999; Pantelis et al., 2002; Petersen and Dixon, 2002a; Sidborn et al., 2003). Although the effects of both particle scale and bulk scale phenomena are important in heap bioleaching, little has been done to systematically integrate particle scale models into bulk scale models. Most existing bulk scale models account for particle scale effects using simplified models.

The focus of this thesis is therefore on providing a systematic link between particle and bulk scale heap bioleaching models by modelling the heap at an intermediate scale. This intermediate scale model is what is referred to as the agglomerate model.

1.4 Thesis Scope and Outline

The aim of this work is to develop an intermediate-scale agglomerate model. The agglomerate model includes particle scale effects, and can be integrated into, or form the building blocks of, a bulk scale model (Fig.1.3). The agglomerate model will be applied to study copper bioleaching from a theoretical case study ore, consisting mainly of chalcocite and pyrite, in the presence of iron oxidizing microbes.

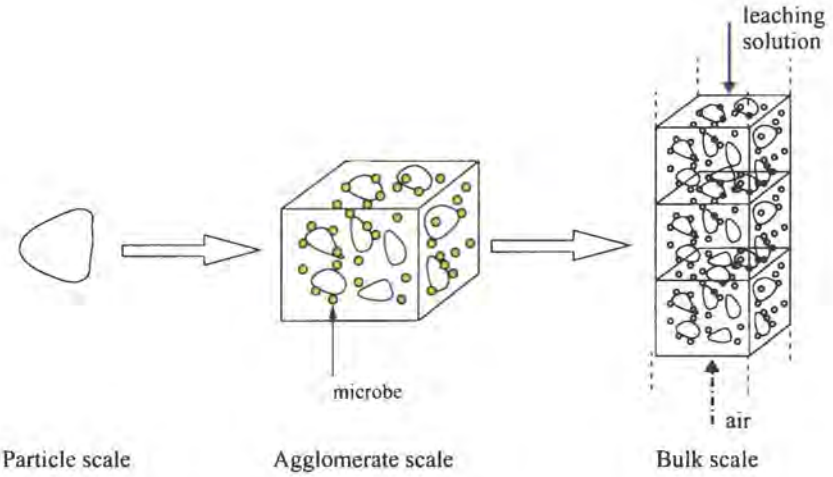


Figure 1.3: Schematic representation of the agglomerate model as an intermediate-scale model.

The thesis is structured as follows. We start by reviewing the approaches in literature to modelling the important sub-processes in heap bioleaching, in Chapter 2. Chapter 3 contains a detailed description of the proposed agglomerate model. In Chapter 4, the agglomerate model is applied to the case study, and the sensitivity of the model, in terms of the effect of various parameters on copper recovery, is investigated. The concluding discussion, and recommendations for further work, are presented in Chapter 5.

CHAPTER 2

Literature Review

This chapter presents a review of modelling strategies employed in literature to describe the main sub-processes that take place in heap bioleaching, especially heap bioleaching of copper sulfide ores. Dump leaching models have also been also been included in this review, as dumps and heaps are based on similar principles. The major difference between the two techniques is that heaps are designed to be more efficient and better controlled; for instance while dumps are aerated only by natural convection, forced aeration is applied in heap bioleaching.

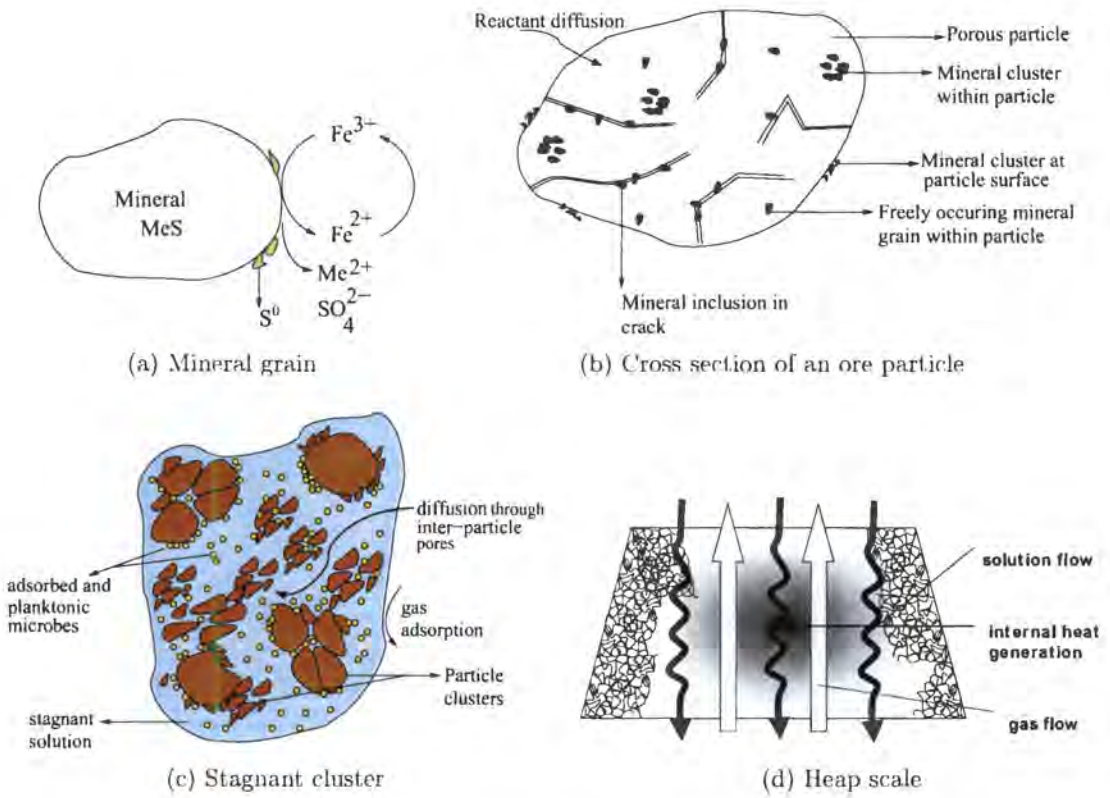
For further reference on a review and comparison of heap leach models, see (Dixon, 2003).

2.1 Overview of Sub-Processes

Bacterial leaching of a copper sulfide ore heap is a complex process that involves the following phenomena:

1. mineral oxidation by reaction with ferric ions.
2. ferrous oxidation (in the presence of microbes, oxygen and acid) to produce ferric, and sulfur oxidation (in the presence of microbes and oxygen) to produce sulfuric acid.
3. transport of the dissolved species in the heap.
4. transport of gas and leaching solution in the heap.
5. transport, growth, attachment and catalytic actions of micro-organisms.
6. generation and transport of heat.
7. hydrolysis and precipitation of ferric based complex compounds.

The phenomena listed above can be classified into micro-scale, meso-scale and macro-scale processes (Dixon and Petersen, 2003).



(a) Mineral grain

(b) Cross section of an ore particle

(c) Stagnant cluster

(d) Heap scale

Figure 2.1: Different modelling scales in heap bioleach modelling.

2.1.1 Micro-Scale Processes

Micro-scale processes occur at the mineral grain and ore particle levels. At the mineral grain level (Fig.2.1(a)), the leaching chemistry and reaction kinetics are the dominant factors. The key mineral leaching reaction in bioleaching of a sulfide mineral is the oxidation of the metal sulfide (MeS) by ferric ions:

$$MeS + 2Fe^{3+} \rightarrow Me^{2+} + 2Fe^{2+} + S^0 \tag{2.1}$$

The chemical reactions are primarily a function of temperature (characterized by the activation energy), and concentration of reactants. Although the principal mechanisms of such reactions are understood, the exact values of critical parameters for each specific case are subject to measurement (Dixon and Petersen, 2003).

At the level of an ore particle (Fig.2.1(b)), the distribution of mineral grains in the particle comes into consideration. The assumption that the mineral grains are uniformly distributed in the ore particle, coupled with relatively fast mineral dissolution, lends itself

to the shrinking core kinetics model (Levenspiel, 1972). In reality, the minerals may occur as free grains or in grain clusters, and may be distributed on the particle surface or as inclusions in veinlets within the ore.

Another important process at the particle level is the transport of reactants to, and reaction products from, reaction sites within the particle. This process is diffusion governed, and is limited by the size and porosity of the ore particle, the diffusion gradient, and the diffusivity of the species.

2.1.2 Meso-Scale Processes

Meso-scale processes occur at the level of a cluster of ore particles (Fig.2.1(c)). The important processes at this level are oxygen uptake into solution, diffusion of dissolved chemical species through the inter-particle pores, and microbial processes.

Oxygen is a key reactant in heap bioleaching as the microbes can oxidize ferrous and sulfur only to the extent to which oxygen is available in the system. Oxygen uptake into solution is a temperature dependent mass transfer step. The mass transfer coefficient is subject to measurement, and has been highlighted as an important parameter in heap leach modelling (Bouffard and Dixon, 2003).

The dissolved chemical species (reactants and reaction products), must diffuse through stagnant solution occupying the bed voidage in order to be recovered in the leaching solution, or to take part in chemical reactions within the ore particles. The extent of the effect of this inter-particle pore diffusion on extraction rate and mineral leaching depends on the length of the diffusion pathway, which may be significant for systems with poor solution distribution (Dixon and Petersen, 2003).

Growth, transport and oxidation of micro-organisms present an additional level of complexity. The key microbial oxidation reactions in sulfide mineral leaching are:

$$4Fe^{2+} + O_2 + 4H^+ \xrightarrow{\text{microbes}} 4Fe^{3+} + 2H_2O \tag{2.2}$$

$$S^0 + 1.5O_2 + H_2O \xrightarrow{\text{microbes}} H_2SO_4 \tag{2.3}$$

Micro-organisms in the heap may be attached to the ore particles or floating in solution. Their growth depends on a number of factors, including availability of nutrients, availability of dissolved oxygen, and temperature.

2.1.3 Macro-Scale Processes

Macro-scale processes are essentially the "flow" processes in the heap. The processes are solution flow, gas flow and heat flow (Fig.2.1(d)).

Heaps need to be irrigated for reagent delivery, product recovery, temperature control, and in order to maintain internal moisture. Heap irrigation is usually by means of sprinklers or drippers, so as to promote uniform solution distribution. Solution flow is a function of heap permeability and saturation. Although a common modelling assumption is uniform solution flow, the flow is usually confined to preferential channels in the heap (Orr, 2002).

Gas transport is important for the supply of oxygen to the heap. Gas transport in a heap is a function of heap permeability and saturation, heap temperature, oxygen depletion and water vapor production. When forced aeration is applied, the effect of gas diffusion on overall gas transport may be reduced (Dixon, 2003).

Heat generated by chemical reactions in a heap is transported via convection by the gaseous and liquid phases, and via conduction through the heap. Heaps assume temperature profiles depending on the irrigation and aeration rates, and these rates can be used to achieve a certain level of temperature control (Dixon, 2000).

In the sections below, approaches in literature to modelling the sub-processes presented above are reviewed.

2.2 Leaching Kinetics

Copper oxide mineral leaching is rapid compared to copper sulfide mineral leaching at ambient conditions. Hence, while the leaching of copper oxide minerals can be represented by a model that assumes diffusion limitation alone, it is important to consider both diffusion and reaction limitation in modelling copper sulfide leaching. Bartlett (1992b) identified two approaches to particle scale modelling for (primary) copper sulfide ore leaching: the pseudo-steady state mixed kinetics model (or reaction zone model) and the unsteady state mixed kinetics model.

The reaction zone model was developed by Braun et al. (1974). In this model, it is assumed that a narrow reaction zone in the ore particle moves topologically inwards during the course of leaching (see Fig.2.2). Steady state diffusion occurs through the

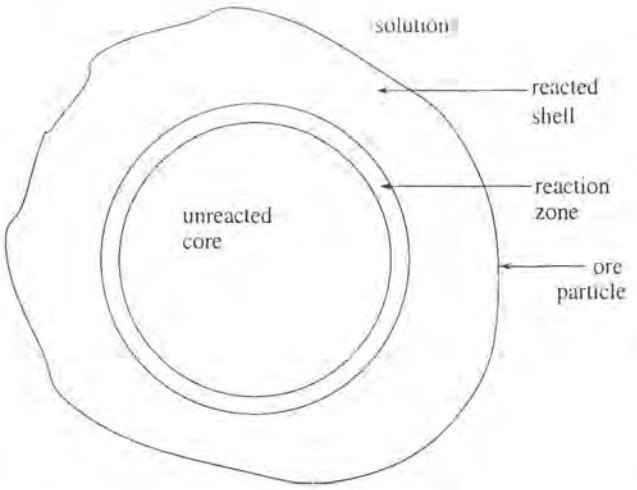


Figure 2.2: Reaction zone model after Braun et al. (1974)

reacted outer region, and the diffusing flux is equal to the rate of chemical reaction within the reaction zone. Mathematically (Bartlett, 1992b),

$$0 = D_e \left(\frac{\partial^2 C}{\partial r^2} + \frac{2}{r} \frac{\partial C}{\partial r} \right) + S \quad . \tag{2.4}$$

Here, C is the species concentration, S is the rate of production or consumption of the species, $D_e = D\epsilon/\tau$ is the effective species diffusivity, where ϵ the particle porosity, and τ the particle tortuosity, a factor that accounts for the windedness of the effective diffusion pathway within the particle. S is obtained by examining each leaching reaction the species takes part in. It is common practice to express the rate of mineral leaching in terms of the rate of mineral conversion X. The rate of mineral conversion within the narrow reaction zone is given by (Braun et al., 1974):

$$\frac{dX}{dt} = \frac{3(1-X)^{\frac{2}{3}} C_{ox}}{\tau_1 + \tau_2 (1-X)^{\frac{1}{3}} (1-(1-X)^{\frac{1}{3}})} , \tag{2.5}$$

where C_{ox} is the concentration of the reactant (ferric for copper sulfide ores). τ_1 and τ_2 are factors that include the grade of the mineral, the intrinsic rate of oxidation, the particle shape factor, the stoichiometric coefficient, the diffusion coefficient of the oxidant, the molecular weight of the sulfide mineral and the radius of the particle.

The more narrow the reaction zone, the closer this model gets to the classical shrinking core model (Levenspiel, 1972). The reaction zone model presented above was applied by Braun et al. (1974 in a modified form that accounts for the occurrence of a higher mineral

grade at the surface of an ore particle, and an increase in the effective reacting interfacial area due to the development of cracks and fissures as leaching progresses. The model has also been modified to account for mass transfer at the liquid-solid film (Sidborn et al., 2003).

The unsteady state mixed kinetics model is stated mathematically below:

$$\epsilon \frac{\partial C}{\partial t} = D_e \left(\frac{\partial^2 C}{\partial r^2} + \frac{2}{r} \frac{\partial C}{\partial r} \right) + S \quad . \tag{2.6}$$

The source term S in Eq.(2.6) is described by a rate law that includes functions for the effect of temperature, solution composition and mineral topology. The exact form of the rate law depends on the mineral being leached. Topological effects in heap bioleaching are not easily determined. This refers to the way in which mineral grains occur in a particular ore, and how they are distributed across different particle size classes. In general, the distribution of mineral grains in an ore particle and across different particle sizes is not homogeneous (Dixon and Petersen, 2003). A certain degree of empiricism is often required to reduce this topological complexity to simple model parameters.

A slightly different approach to modelling leaching kinetics was taken by Casas et al. (1998), Moreno et al. (1999) and Leahy et al. (2003). In these models, the rate of copper extraction was defined in terms of microbial activity alone using the Michaelis-Menten equation, with dissolved oxygen as the limiting reactant. This model is a steady state simplification of Eq.(2.6), ignoring diffusion effects and assuming the source term S is purely due to microbial activity.

In addition to copper mineral dissolution reactions, other chemical reactions may have a significant impact on the overall leaching process. Examples of such chemical reactions are acid consumption by gangue, and equilibrium processes such as precipitation, complexation, and adsorption. These can affect reagent and product balances, or induce pore plugging and loss of permeability (Dixon, 2003). For instance, a concern in the ferric leaching of chalcopyrite is the passivation of the mineral at high solution potentials. Although formation of elemental sulfur layers and surface precipitation of jarosites have been offered as an explanation, it is still unclear why this phenomenon occurs (Olson et al., 2003; Petersen and Dixon, 2002b).

2.3 Biological Phenomena

The use of microbes in the solubilization of metals has deep historical roots dated as far back as the fifteenth century (Olson et al., 2003; Rawlings, 2002). The action of microbes in industrial metal recovery serves one of two purposes. One is their use in the pre-treatment process to oxidize unwanted minerals, thereby permitting other chemicals to penetrate the ore and liberate the desired metal. This process is known as biooxidation, and is applied in the removal of sulfide minerals from refractory gold ores. The other action of microbes is in the conversion of insoluble metal sulfides to water soluble sulfates. This process is known as bioleaching. The bioconversion of metal sulfides to metal sulfates is also the major cause of acid-mine drainage. Other microbial species have found application in its treatment, (where they are used to oxidize and precipitate iron), and also in the desulfurization of coal (Nemati et al., 1998).

Micro-organisms, specifically iron and sulfur oxidizing microbes, play the important role of ferric and acid regeneration in heap bioleaching operations, thus producing *in situ* the reagents needed for mineral oxidation. A large and growing body of literature exists on the microbial leaching process (for reviews see Hansford, 1997; Nemati et al., 1998; Ojumu et al., 2005; Rawlings, 2002, 2005). The objectives of these studies are to provide a fundamental understanding of the microbial oxidation mechanisms, to gain insight into the identification of the microbial strains best suited to leaching particular types of ore, the best operating conditions for the bacteria, and the effect of diverse conditions and substances on these microbes.

Two mechanisms have been proposed in literature for the microbial oxidation of minerals in heap bioleaching. These are the direct and indirect mechanisms. In the direct mechanism, the mineral is leached by some biological agent, in other words the micro-organisms obtain electrons directly from the mineral (Silverman and Ehrlich, 1964). The bacteria are attached to the ore, which brings them in close proximity with the mineral. It is now generally accepted that the direct mechanism does not exist for the ferric oxidation of minerals (Rohwerder et al., 2003; Tributsch, 1999), although it may exist for elemental sulfur oxidation. In the indirect mechanism, the minerals are leached only by ferric ions, and the role of the microbes is to oxidize the ferrous ions formed in the mineral leaching process back to ferric. This mechanism is further divided into contact and non-contact mechanisms (Rawlings, 2002). The non-contact mechanism is the oxidation of ferric ions in solution by planktonic microbes. The contact mechanism takes into account that most

cells attach to the surface of sulfide minerals, and by means of ferric ions complexed in their extracellular polymeric substances (EPS) layer, they begin to degrade the sulfide minerals (Sand and Gehrke, 2006). This means that the electrochemical processes resulting in the dissolution of sulfide minerals take place at the interface between the bacterial cell (wall) and the mineral sulfide surface (Rohwerder et al., 2003). The microbes typically form an extracellular polymeric substances (EPS) layer when they adhere to the surface of the mineral, and it is within this layer that the microbial oxidation reactions take place most rapidly and efficiently (Rawlings, 2005).

The micro-organisms that have been identified in bioleaching fall into mesophiles (operating at 15–45℃), thermophiles (operating at 40–60℃) and extreme thermophiles (operating at 50–90 ℃). The prominent mesophiles in bioleaching are *Acidithiobacillus ferrooxidans*, *Acidithiobacillus thiooxidans*, *Acidithiobacillus caldus*, *Leptospirillum ferrooxidans* and *Leptospirillum ferriphilum* (Rawlings, 2005). Examples of moderately thermophilic bacteria are *Acidithiobacillus caldus* and *Sulfobacillus thermosulfidooxidans*, and extreme thermophiles are mostly archaea. Four important characteristics of these micro-organisms are (Rawlings, 2005):

1. They grow autotrophically by fixing carbon dioxide from air.
2. They get their energy by using ferrous ion or reduced inorganic sulfur compounds as electron donor, and oxygen as electron acceptor.
3. They are acidophiles – they grow at pH of typically 1.4–1.6.
4. They are tolerant to a wide range of metal ions.

In order to include biological processes in a mathematical model of heap bioleaching, microbial growth, oxidation and transport must be expressed mathematically. The development of a comprehensive mathematical description of the rate of microbial oxidation (which incorporates the effects of substrate concentration, temperature, pH, biomass concentration, ionic strength, and other factors) is an area of intense research. Both empirical and qualitative functions for microbial oxidation have been implemented in heap bioleaching models.

The effect of microbial oxidation alone on extraction (excluding microbial growth and transport), can be studied by assuming a constant concentration of micro-organisms in a heap. This is the approach taken by Pantelis and Ritchie (1992), Casas et al. (1998) and Sidborn et al (2003). Pantelis and Ritchie (1992) incorporate microbial effects by

implementing a temperature function that multiplies the shrinking core equation. This temperature function is 1 for temperatures less than a certain threshold T_0, decreases monotonically to zero for $T_0 < T \leq T_{00}$, and is zero for $T > T_{00}$. This disregards the actual microbial oxidation reaction, and hence has a limited scope.

Casas et al. (1998) assume that the ferrous ions are always in excess, and so express the rate of copper sulfide dissolution purely as a function of the rate at which the microorganisms consume dissolved oxygen. This function is implemented as a Michaelis-Menten equation. The dependence of microbial respiration rate on temperature was expressed by fitting a model equation to experimental data for the ferrous oxidation of *At. ferrooxidans*. This model clearly overlooks the kinetic dependence of copper sulfide leaching on ferric ion concentration. Sidborn et al. (2003) also express the rate of oxygen consumption in the heap by a Michaelis-Menten equation. However, this is related to the rate of ferric regeneration in the heap. Again, ferrous ions are assumed to be in excess in the bulk solution, and so the rate of ferric regeneration is limited only by the availability of dissolved oxygen. Mineral oxidation is modelled separately using the reaction zone model with ferric ions as the oxidant.

The models by Neuburg et al. (1991), Dixon and Petersen (2003) and Leahy et al. (2005a) account for both microbial growth and microbial transport. The model by Neuburg et al. (1991) employs Monod kinetics to describe the effect of ferrous ion on microbial growth. Also accounted for are the effects of pH and dissolved oxygen, although the form of this dependence is not explicitly stated. The model does not, however, explicitly include a death rate for the microbes. The authors distinguish between adsorbed and desorbed microbial populations, and include the rate of adsorption and desorption in the model. The rate of ferrous consumption is related to the microbial kinetics through the microbial yield parameter in the standard way. Advective microbial transport is also considered.

The model by Leahy et al. (2005a) takes a similar, but more sophisticated approach. The maximum microbial growth rate is modified by the effects of temperature (using the empirical model by Casas et al. (1998)), and Monod terms for dissolved oxygen and ferrous ions. A specific death rate of the microbes is included in the model. Adsorbed and desorbed microbes are also considered separately. The rate of adsorption and desorption is accounted for following a model by Tan et al. (1994). No differentiation is made between bulk and particle concentrations of the dissolved chemical species, and so the net rate of change of ferrous ion is obtained by subtracting the rate of consumption by microbes

(obtained via the yield coefficient) from the rate of production in the chemical leaching reactions. Advective and diffusive transport of the desorbed microbes is also considered.

The model by Dixon and Petersen (2003) is currently the only one (among the models reviewed here) that includes sulfur oxidizing microbes. In addition to the limitation to microbial growth rate caused by ferrous and oxygen, this model allows for the influence of acid concentration, and growth inhibition due to overcrowding. The model also includes effects that characterize the microbial population under stress, such as cell maintenance and endogenous decay. The authors differentiate between a growth temperature function that affects the microbial growth kinetics, and a death temperature function which accelerates microbial death rate at unfavorable temperatures. Microbial adsorption and desorption is accounted for by the Langmuir adsorption isotherm. The microbial transport model consists of a diffusion-reaction equation for the microbes in the stagnant channels, and an advection equation for the flowing channels.

The microbial models have been presented here in an approximately increasing order of sophistication. It is observed that while it is desirable to implement a comprehensive microbial model, the number of parameters increase with increasing level of sophistication, making the model characterization more difficult. Hence, there is a trade-off between the increased sophistication, and the ease of implementation of the microbial model.

2.4 Solute and Solution Transport

Chemical and microbial species in a heap are transported by diffusion through the solution filled spaces, and by the advection of the leaching solution. Solution flow in a heap can be described by the theory of unsaturated fluid flow in porous media.

The transport equation for a solute i in the liquid phase takes the form:

$$\frac{\partial(\varepsilon_{\mathrm{L}} C_i)}{\partial t} + \nabla \cdot (\varepsilon_{\mathrm{L}} C_i \mathbf{u}_{\mathrm{L}} - D_i \nabla(\varepsilon_{\mathrm{L}} C_i)) + S_i = 0 \,. \tag{2.7}$$

C_i is the liquid concentration of solute i, ε_{L} is the liquid volume fraction of the ore bed, $\mathbf{u}_{\mathrm{L}}$ is the liquid velocity, D_i is the diffusion coefficient of the solute in the liquid, and S_i is the rate of consumption or generation of the solute. There are variations to the implementation of Eq.(2.7) in heap bioleaching literature. For instance, Eq.(2.7) may be modified to include terms for adsorption (Bouffard and Dixon, 2003; Cariaga et al., 2005; Dixon and Petersen, 2003).

It is common practice to calculate the liquid velocity $\mathbf{u}_\mathrm{L}$ from Darcy's law:

$$\mathbf{u}_\mathrm{L} = -\frac{K_\mathrm{in} k_\mathrm{L}(s)}{\varepsilon_\mathrm{L}\mu_\mathrm{L}}(\nabla p_\mathrm{L} + \rho_\mathrm{L} g \nabla z) \quad , \tag{2.8}$$

where K_in is the intrinsic permeability of the heap, s is the effective saturation of the heap, k_L is the relative permeability of the wetting phase (water in this case) and depends on s, μ_L, ρ_L and p_L are the viscosity, density and pressure of the flowing liquid. The pressure of the flowing liquid depends on capillary pressure, which depends in turn on the saturation of the heap.

The mass conservation equation for unsaturated flow of solution through a porous medium is:

$$\frac{\partial(\varepsilon_\mathrm{L}\rho_\mathrm{L})}{\partial t} + \nabla \cdot (\rho_\mathrm{L}\mathbf{u}_\mathrm{L}) = S \, . \tag{2.9}$$

Here, S is the volumetric production or consumption of the liquid. The liquid velocity $\mathbf{u}_\mathrm{L}$ is given in Eq.(2.8). Substitution of Eq.(2.8) in Eq.(2.9) gives rise to Richard's equation, which is an advection-diffusion equation for the effective saturation of the heap. Solution flow tends to follow preferential flow paths due to uneven heap permeability and saturation. This may result in flowing channels that are separated by large pockets of stagnant solution.

Most heap bioleaching models consider only the one-dimensional downward transport of solutes in a heap under conditions of constant liquid velocity (Dixon and Petersen, 2003; Leahy et al., 2005a; Sidborn et al., 2003). The effect of heap saturation has also been investigated by some authors (Bennet et al., 2003; Cariaga et al., 2005; Sheikhzadeh et al., 2005). Cariaga et al. (2005) developed a model that simultaneously considers the change in bed saturation and the transport of two components (copper and acid) by the fluid phase in heap leaching. Dixon and Petersen (Dixon and Petersen, 2003; Petersen and Dixon, 2002a) attempt to capture the segregation of solution in a heap into flowing and stagnant regions by dividing the heap into bulk flow channels surrounded by stagnant diffusion zones. The stagnant zones are conceptually defined as side branches attached to the bulk flow channel, and one dimensional transverse solute diffusion is modelled here. Transport in the bulk flow channel is modelled by the one dimensional advection equation in the axial direction, with a source term for the contribution to the solute concentration from the diffusion side branches. This formulation captures transverse diffusion which can be very important, especially in situations where flowing channels are separated by large pockets of stagnant solution (Petersen and Dixon, 2003).

2.5 Gas Transport

Gas (air) transport is an important aspect of heap bioleaching, as oxygen availability is crucial to the oxidation reactions occurring in a heap. Natural convection is typically the only means of air transport in dumps; heap leaching on the other hand involves the forced aeration of heaps. Natural convection in a dump or heap is due to variations in gas density, caused by temperature changes, oxygen depletion and humidification (Casas et al., 1998).

As stated above, the component of interest in the gas phase with respect to heap (and dump) leaching is oxygen, even though some models include the transport of water vapor by the gas phase (see Pantelis et al., 2002). Although carbon dioxide is also an important substrate for the microbes, none of the models that are reviewed here consider it. The transport of oxygen in the gas phase is given by Eq.(2.7) where liquid phase parameters are replaced by gas phase parameters. Even though most models consider the axial diffusion of oxygen, (which is important in dump leaching), its relative magnitude is insignificant when forced aeration is employed (Dixon, 2003).

Gas flow in a porous medium depends on the degree of water saturation. When the water saturation is relatively small, the gas phase is continuous and can flow through the medium. A high water saturation, on the other hand, results in the gas phase becoming disconnected and occluded (Fredlund and Rahardjo, 1993; Pantelis et al., 2002). The mass balance for the gas phase is given by Eq.(2.9), replacing all subscripts 'L' with 'g' to represent the gas phase. Most authors calculate the gas phase velocity from Darcy's law (see Eq.(2.8)), which takes the degree of water saturation into account by including the relative gas permeability of the bed $k_g(s)$ (in place of $k_L(s)$ in Eq.(2.8)), where s is the degree of water saturation.

Casas et al. (1998) and Sidborn et al. (2003) reported that under natural convection conditions, air flows into the heap from the sloped sides and out at the top. This leads to oxidation rates that are higher at the bed slopes and decrease as we move inwards. Both authors assume constant bed permeability. Moreno et al. (1999) extended the model by Casas et al. (1998) to include a varying bed permeability that increases continuously from top to bottom. They argue that the segregation of particles of different sizes when the bed is being built leads to the accumulation of coarse ore particles at the bottom of the bed. It seems more likely however, that the materials at the bottom of the heap will be compacted as more material is added to the top of the heap, causing heap permeability

to decrease with depth. Therefore, this argument is questionable. They reported higher oxygen concentration towards the center of the bed, leading to higher copper recovery.

Forced aeration has the effect of increasing copper extraction, as reported by Leahy et al. (2003). The authors accounted for forced aeration by solving steady state Navier-Stokes equations. They expressed the body force in these equations in terms of the porous media resistivity to flow, and the effect of gravity. Their simulation results for the case studied also indicated that for heap permeabilities greater than a certain threshold, better copper extraction was not achieved by forced aeration. The model was later extended by solving unsteady-state Navier-Stokes equations (Leahy et al., 2005b). However, a different heap geometry was implemented in this case. The authors reported that simulation results showed a bottom-up trend in copper leaching, because the iron oxidizing microbes were initially limited by low oxygen concentrations at the top of the heap.

Dixon and Petersen (2003) assumed that forced aeration led to a negligible effect in the axial diffusion of oxygen. A steady-state was also assumed, such that the rate of oxygen transfer between the gas phase and the liquid phase was equal to the rate of oxygen consumption in chemical reactions within the heap. Oxygen transport was implemented as a one-dimensional advection equation. The authors applied this model to both heap and column leaching of an ore containing chalcocite and pyrite, and found that in a situation where oxygen was in abundant supply relative to demand, increasing the aeration rate had no effect on copper extraction, and the rates of heat generation were insufficient for the rate of aeration to have much influence on the temperature distribution. This model was also applied by Bouffard and Dixon (2003) in studying the biooxidation of pyritic refractory gold under isothermal conditions. They reported that at high temperatures (> 45°C), the oxygen gas-liquid mass transfer seemed to be the rate-limiting step.

2.6 Heat Transport

Heat transport in a heap is important for several reasons. Firstly, the micro-organisms which catalyze the oxidation reactions are sensitive to temperature. As stated in Section 2.3, these micro-organisms are grouped into mesophiles, moderate thermophiles and extreme thermophiles depending on the temperature ranges within which they exist. It might be advantageous to maintain a heap within a certain temperature range to ensure the survival of a particular strain of micro-organisms. This is useful, for instance, in the bioleaching of chalcopyrite, which leaches to higher extractions using thermophiles in the

temperature range of 65–75 °C, compared to its leaching using mesophiles (the mineral has a tendency to passivate at the high solution potentials associated with mesophile leaching) (Petersen and Dixon, 2002b). Secondly, many oxidation reactions central to copper recovery from secondary ores are strongly temperature dependent. For instance, the reaction for the leaching of covellite has a high activation energy, hence every 10°C increase in temperature represents an almost threefold increase in the reaction rate (Dixon, 2000).

Heat is transported in a heap by advection of the liquid and gas phases, and by conduction. This can be represented by an equation similar to Eq.(2.7):

$$\overline{\rho c_{\mathrm{p}}} \frac{\partial T}{\partial t} + \nabla \cdot \left(T\left[(\rho_{\mathrm{L}} c_{\mathrm{pL}} \mathbf{u}_{\mathrm{L}}) + (\rho_{\mathrm{g}} c_{\mathrm{pg}} \mathbf{u}_{\mathrm{g}})\right] - (\overline{k_{\mathrm{B}}}\, \nabla T) \right) + Q = 0 \,, \tag{2.10}$$

where $\overline{\rho}, \overline{c_{\mathrm{p}}}$ and $\overline{k_{\mathrm{B}}}$ are the average heap density, heat capacity and thermal conductivity respectively, the subscripts 'L' and 'g' stand for the liquid and gas phase respectively, and Q represents heat generation (or consumption) in the heap. Q incorporates the heat of reaction, heat of solution of reaction products, and heat of evaporation and condensation.

Several heap bioleaching models include a model for heat balance (see Bennet et al., 2003; Leahy et al., 2003, 2005a; Pantelis and Ritchie, 1992; Sidborn et al., 2003). Dixon (2000) investigated heat conservation in heap leaching by developing a model that accounts for such boundary effects as evaporation, convection and radiation at the heap surface (based on geo-climatic conditions), and heat exchange at the point of application of process air to the lower heap surface. The model also accounts for evaporation of water within the heap, and hence treats both heat carried by dry air, and heat carried by water vapor. For the purpose of the study, a constant rate of heat generation in the heap by chemical reactions was assumed, and the parameters were set up to simulate a heap operating in a desert, and at high altitude. By keeping all other parameters constant and varying the ratio of the aeration rate (G_a [kg/m^{2}h]) to the solution application rate (G_L [kg/m^{2}h]), two distinct regimes were observed in the model simulation results, with a critical G_a/G_L where the regimes switch. The first regime had, at steady state, low temperatures at the top of the heap and high temperatures at the bottom of the heap. The other regime had high temperatures at the top of the heap and low temperatures at the bottom of the heap. A given steady state regime of the heap was attained for a given G_a/G_L ratio, regardless of the initial temperature of the heap. As evaporation at the heap surface is a critical factor for a heap operating under desert conditions, the effect of

an evaporation shield on the heap heat conservation was investigated. The author also investigated the effects of solution and air heating on the heat profile in a heap. The conclusion drawn from these investigations was that the most effective way of achieving a high degree of heat conservation for a heap in the high desert is by controlling the G_a/G_L ratio.

Although the model described above provides insight into predicting heat conservation in a heap, its main limitation with respect to application is that it is developed in isolation from the chemical and microbial processes occurring in the heap (a constant rate of heat generation in the heap was assumed). Hence if integrated with a model for solute and microbial kinetics, and given values for the volumetric heat generation of specific minerals, the model may provide more reliable information on the effect of operation parameters (such as G_a and G_L) on heap performance. This was done by Petersen and Dixon (2002b) in investigating the thermophilic heap leaching of a chalcopyrite concentrate. Geo-climatic conditions were chosen to reflect conditions at an existing mine site. They observed that for the case studied, varying the G_a/G_L ratio showed a steady state temperature in the 60–85 °C range. Temperature profiles were such that a peak developed near the top of the heap (environmental effects led to a reduction in temperature at the heap surface), and the heap was heated homogeneously within the active zone of the heap. Substantial cooling took place only in the drainage layer, with the outflowing solution more or less at the same temperature with the inflowing. Different heap heights were also modelled, indicating that shorter heaps (3.5m active zone) allow more favorable temperature distributions than taller heaps, but are otherwise equivalent.

Leahy et al. (2005a) studied microbial temperature dependence in heap bioleaching of chalcocite by solving Eq.(2.10) in conjunction with equations for mineral leaching, solute transport and microbial oxidation and transport. The average heap parameters $\overline{\rho}, \overline{c_p}$ and $\overline{k_B}$ in Eq.(2.10) were calculated from weighted sum formulae involving the solid, liquid and gas phases, and the source term Q was calculated from the heats of reaction. The effective heat generation in the heap, obtained by considering only the most exothermic and endothermic reactions, was exothermic. Constant temperature boundary conditions were assumed. The heat model implemented here is, on the whole, less sophisticated than that developed by Dixon (2000); it does not include the boundary effects stated in the previous paragraph, or the effect of water evaporation within the heap. The simulations were set up under ferric limiting conditions, such that the rate of leaching depended on the rate of microbial production of ferric. The effect of temperature on microbial

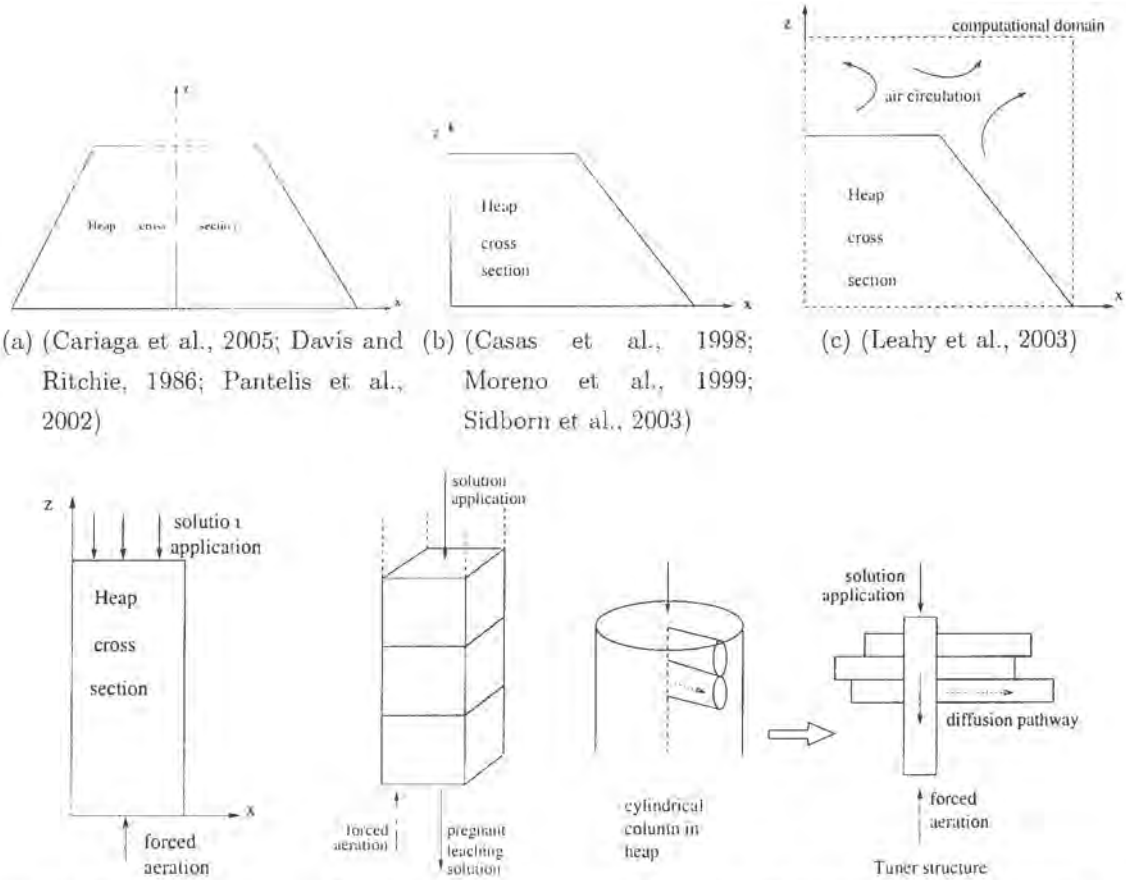
growth rate was implemented using an empirical function. Simulation results showed a top-down leaching mechanism in the presence of mesophiles, with the development of a sharp front within the heap. A number of factors were investigated, including the effect of solution flowrate, and initial heap temperature. Findings indicated that the rate of mineral leaching was controlled by the rate of ferric regeneration by microbes, and this rate in turn was strongly dependent on the temperature profile within the heap. The temperature profile developed as an interplay between the cooling effect on incoming liquid, and the heat generated by mineral oxidation in the heap.

As the effect of air flow rate with respect to heat conservation was not considered by Leahy et al. (2005a), an interesting investigation would be the effect of the G_a/G_L ratio on copper extraction. Varying this ratio may change the top-down leaching profile observed by Leahy et al., and may lead to entirely different profiles for the two heat regimes discussed by Dixon.

2.7 Heap Geometry

A heap may measure from several meters to several tens of meters in height and several hundreds of meters in width. Most heaps are built in 6–10m lifts; either single lifts (pad leaching, each lift is removed at the end of leaching), or multiple lifts (each lift built on top of existing lifts). Heaps built into valleys can be several hundred meters in height. Although the specific geometry of a heap may be difficult to determine, a domain must be specified for the equations that model a heap. Most heap models in literature are developed within the context of a two-dimensional heap cross section.

A popular assumption in literature is that the cross-section of a heap is trapezoidal in shape, and the model is solved for the full trapezium (Cariaga et al., 2005; Davis and Ritchie, 1986; Pantelis et al., 2002) (see Fig.2.3(a)) or half trapezium (Casas et al., 1998; Leahy et al., 2003; Moreno et al., 1999; Sidborn et al., 2003) (see Fig.2.3(b), Fig.2.3(c)). Other geometries in literature include cylindrical (Bouffard and Dixon, 2003; Pantelis and Ritchie, 1992; Petersen and Dixon, 2002a), truncated cone (Pantelis and Ritchie, 1991), rectangular cross-section (Leahy et al., 2005a,b) (see Fig.2.3(d)), and a three dimensional approach by assuming a stack of unit volumes (Bartlett, 1992b; Roman and Olsen, 1974) (see Fig.2.3(e)). Trapezoidal type heaps geometries (Fig.2.3(a)–Fig.2.3(c)) are relevant only in the context of free convection models, where air flow at the heap slopes may be significant. Otherwise, the sloped sides of the heap are insignificant compared to the



(a) (Cariaga et al., 2005; Davis and Ritchie, 1986; Pantelis et al., 2002)

(b) (Casas et al., 1998; Moreno et al., 1999; Sidborn et al., 2003)

(c) (Leahy et al., 2003)

(d) (Leahy et al., 2005a,b)

(e) (Bartlett, 1992b; Roman and Olsen, 1974)

(f) Cylindrical column to Turner structure (Bouffard and Dixon, 2003; Petersen and Dixon, 2002a).

Figure 2.3: Some heap geometries in literature.

typical dimension of a heap. The application of forced air (and leaching solution) at spaced intervals in a heap suggests the assumption that the heap is made up of recurring sections. Bearing in mind that forced aeration reduces the effect of air diffusion within a heap, periodic boundary conditions can be assumed within the heap, and a section can be modelled either as a cylinder or a stack (Fig.2.3(d)–(f)).

The effect of heap geometry on model results was investigated by Pantelis and Ritchie (1992). The authors studied the effect of a cylindrical shaped heap, and truncated cone shaped heap on the dump leaching of pyritic ore, and found that the temperature, oxygen and gas velocity profiles within the heap were different for the different geometries, and

that the amount of material oxidized in the first four years in the truncated cone heap was less than in the cylindrical heap. This suggests that a geometry close to that of the planned heap should be used when making predictions about heap performance from mathematical models.

2.8 Numerics

The mathematical model for heap bioleaching is a set of coupled differential equations, to which an analytical solution is usually extremely difficult to determine. Hence the equations must be solved numerically on a computer. Issues surrounding the numerical solution of differential equations include stability, consistency and convergence. Stability is not an independent quality of a discretization, but depends on the relative size of the spatial and temporal node steps, as well as the actual differential equation being solved. Apart from the question of the consistency of the numerical method being used (which is treated extensively in texts), consistency in the application of the numerical discretization to the analytic equations should also be addressed. In other words, whether the discrete equations, and the solution algorithm implemented, represent the analytic case that we intend to solve.

Some of the numerical methods that have been applied to obtain solutions to heap bioleaching models in literature are (implicit) finite difference methods (Casas et al., 1998; Dixon and Petersen, 2003; Pantelis and Ritchie, 1992; Sheikhzadeh et al., 2005), finite volume methods(Bennet et al., 2003; Cariaga et al., 2005) and finite element methods (Cariaga et al., 2005; Sidborn et al., 2003). Some authors have reported writing their own code (Dixon and Petersen, 2003; Pantelis and Ritchie, 1992), while others have used commercial software (Leahy et al., 2005a,b; Sidborn et al., 2003).

2.9 Conclusion

A review of the sub-processes in heap bioleaching, and the various models that have been used to represent these sub-processes, has been presented. Heap bioleaching models in literature have become increasingly sophisticated, as the sub-phenomena and driving forces in the heap are better understood, and with higher computing power available to solve these sophisticated models. Over time, the emphasis in heap modelling has moved from particle scale phenomena, to bulk scale phenomena. However, this transition does not offer a systematic inclusion of particle scale effects, such as particle size distribution, into the bulk scale model. Often, simplified particle scale models, such as the shrinking core model, with a single ore particle radius, are assumed in the bulk scale models.

Fundamental to the choice of model scale is an assumption of which sub-processes are limiting in a heap. In particle scale models, it could be the reaction kinetics, mass transfer effects, or biokinetics. In the bulk scale models, it is usually the supply of oxygen that is limiting above all else. One aspect that has come out of the work of Dixon and Petersen (Dixon and Petersen, 2003; Petersen, 2006; Petersen and Dixon, 2002b, 2003) is that both local and bulk effects can be rate-limiting, and one cannot predict *a priori* which effect will be rate-limiting. Hence, there is a need for a model that integrates both particle and bulk scale effects. This is the focus of this thesis, and is pursued through the development of an intermediate scale agglomerate model.

CHAPTER 3

Model Development

In the previous chapter, the classification of the physical processes that take place in heap bioleaching into micro-, meso- and macro-scale processes was discussed. The focus of this chapter is on the development of a mathematical model for the meso-scale. This meso-scale model will henceforth be called the "agglomerate model".

In this chapter, the agglomerate model is developed within the context of bioleaching of copper sulfide ores. The model will be built on fundamental principles, and so can be extended to include other copper minerals.

The agglomerate is described in detail in Section 3.1, and the model equations are derived in Section 3.2.

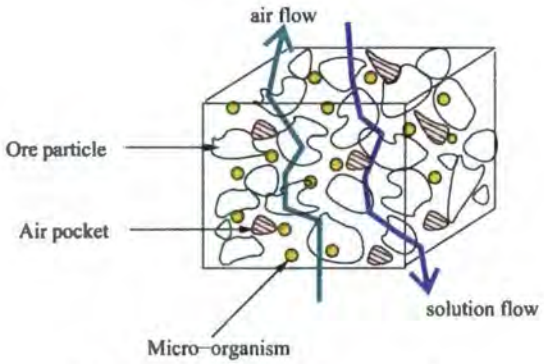
3.1 Problem Formulation

3.1.1 Definition of the Agglomerate

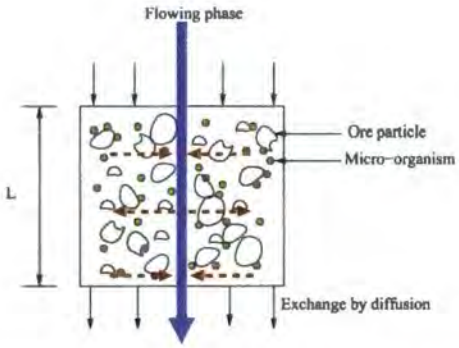
The agglomerate model for heap bioleaching is an intermediate model between a particle scale model and a heap scale model. As such, the agglomerate model includes detailed particle scale processes, and can form the building blocks of (or be incorporated into an existing) heap scale model.

The agglomerate model is developed in the context of a representative unit volume in a heap. The unit volume approach to heap bioleaching was applied in an early model by Roman and Olsen (1974). However this application had limited sophistication, and the concept is driven further here. A conceptual diagram of the agglomerate is shown in Fig.3.1. In this work, the agglomerate is defined as a unit volume of a heap that comprises a solid phase (a size distribution of ore particles), a liquid phase (stagnant and flowing leaching solution, which contains dissolved solutes, attached and planktonic microbes) and a gas phase (flowing air and air pockets). The significant processes that occur at this level are :

- Mineral dissolution reactions.



(a) The agglomerate as a unit volume



(b) 2-D schematic view of agglomerate model. The broken arrows represent exchange between the flowing and stagnant solution

Figure 3.1: The agglomerate model.

- Microbial processes (growth, oxidation and transport).
- Diffusion of dissolved solutes within the ore particles, and through the stagnant solution.
- Heat generation and diffusion.
- Solution and gas flow through the agglomerate.

The agglomerate model can be extended to a heap scale model by stacking the unit volumes to form a unit heap column, and including in the model the interaction between neighboring unit volumes in the unit heap column, and bulk flow phenomena. Further development from this point will require a description of the interaction between adjacent heap columns. The scope of this work is limited to modelling a single agglomerate unit, hence the development of a heap scale model will not be pursued.

3.1.2 Model Description

In this section, an overview of the physical processes that are included in the model, and the assumptions made in model development, are given.

Chemical Species Modelling

The gas phase (air flow and gas pockets) is not explicitly included in the model. It is rather assumed that dissolved oxygen is present in the incoming flowing solution at a given concentration. The liquid phase in the agglomerate is separated into the stagnant (or bulk) and flowing phases, with exchange of chemical and microbial species occurring across these phases. This separation is to capture the effect in heaps where significant lateral flow of solution away from channels following the initial heap wetting is not expected (Dixon, 2003). The change in the bulk concentration of a species is a result of exchange with the flowing solution, and diffusion out of (or into) the ore particle (see Fig.3.1(b)). It is assumed that there is no exchange across the agglomerate faces by diffusion. It is also assumed that the dissolved species are uniformly distributed in the bulk solution at any given time, and so diffusion within the bulk solution is not included in the model. Chemical and microbial reactions in the flowing solution are not included in the model.

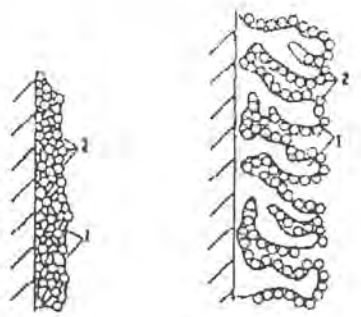
Transport by the flowing solution is modelled by an advection equation. Although the solution may follow several preferential flow patterns (channelling) within the agglomerate, this is not taken into consideration in the model.

Diffusion and reaction of species in the ore particles is modelled by a diffusion-reaction equation. A perfectly spherical shape is assumed for the ore particles. A surface layer surrounding the particles is assumed, such that a diffusion gradient exists between the surface concentration and the bulk concentration of any given chemical species. The resistance to mass transfer across this layer is expressed in terms of a simplified mass transfer coefficient. Although a size distribution of ore particles will be considered, the particles are assumed to be well distributed in the agglomerate, and occupying a certain volume fraction, ε_p, of the agglomerate. Representing the volume fraction of the bulk solution as ε_b, and that of the flowing solution as ε_f, and bearing in mind that the gas phase is not modelled explicitly, then $\varepsilon_p + \varepsilon_b + \varepsilon_f = 1$.

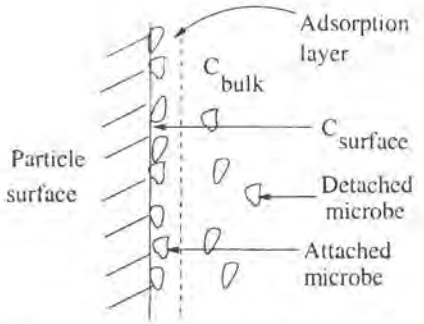
Microbial Modelling

Micro-organisms in the agglomerate may be adsorbed (sessile) onto the ore surface, or desorbed (planktonic). Microbes that are attached to the ore particle surface form an extracellular polymeric substances (EPS) layer. It is proposed that the EPS layer is heavily impregnated with ferric ions, and so provides a reaction space within which the microbial oxidation reactions take place more rapidly (Rawlings, 1997b, 2005; Rohwerder

et al., 2003). Although the microbial attachment process is predominantly mediated by the EPS surrounding the cells, the microbial interfacial processes with regards to the ferric/ferrous cycle are not clarified (Sand and Gehrke, 2006).



(a) Classic biofilm (left) and biofilm with adsorbed cells (right) (Karamanev, 1991; Nemati et al., 1998). "1" points to extracellular polymer c substances (left) and biofilm framework (right). "2" points to the cells.



(b) Schematic representation of present model for EPS layer. C_{bulk} is the concentration of a chemical species in the bulk solution, while $C_{surface}$ is the concentration of a chemical species at the particle surface.

Figure 3.2: Microbial adsorption layer.

Fig.3.2(a) shows two biofilm structures in literature. The structure on the left is the classic biofilm structure, which consists of cells forming the framework of the biofilm and bridges of the EPS layer (Nemati et al., 1998). A different structure was proposed by Karamanev (1991), where the cells do not form part of the mechanical framework of the biofilm. Rather, the cells are adsorbed as a monolayer to the surface of the biofilm, made of jarosites.

Fig.3.2(b) is a schematic representation of the model implemented in this work. Adsorbed microbes are treated differently from desorbed microbes (in terms of their respective growth rates), and the two populations related using the Langmuir isotherm. Microbial activity at the particle surface is accounted for by introducing a microbial source term at the particle surface.

It will be assumed that microbial kinetics in the flowing solution is negligible, and so the flowing solution only serves to transport the microbes between unit volumes. It is also assumed that the planktonic microbes are uniformly distributed in the bulk solution. Changes in the respective adsorbed and desorbed microbial populations in the agglomerate are due to growth/death kinetics, the adsorption/desorption process, and exchange

between the bulk and flowing liquid phases.

Summary of Model Assumptions

1. The agglomerate is saturated with liquid, and the gas phase (consisting of flowing gas and stagnant gas pockets) is not included in the model.
2. There is a fixed concentration of dissolved oxygen in the incoming liquid phase.
3. The leaching solution in the agglomerate is separated into the stagnant and flowing phases.
4. The concentration of chemical and microbial species in the bulk solution is uniform.
5. There is no exchange, by diffusion, of chemical and microbial species across the agglomerate boundaries.
6. The ore particles are spherical and contain uniformly distributed minerals at initial time.
7. Diffusion of dissolved solutes within the ore particles proceeds principally in the radial direction, hence angular diffusion is neglected.
8. Chemical and microbial reactions in the flowing solution are not considered – only transport by the flowing solution is considered.
9. The growth rates of adsorbed and desorbed microbes (for a given species) are treated differently.

3.2 Model Equations

3.2.1 Transport Model

Chemical Species

Each particle size class j is modelled by a representative (or mean) particle of radius R_j. The mass balance equation for each dissolved species i, in each size class j, is given by the diffusion-reaction equation below:

$$\left[\epsilon\frac{\partial C_i}{\partial t} = \epsilon D_i\left(\frac{\partial^2 C_i}{\partial r^2} + \frac{2}{r}\frac{\partial C_i}{\partial r}\right) + \sum_l \nu_{il} S_l + \delta(R-r)\sum_m \nu_{im} S_{\mathrm{surf},m}\right]_j , \qquad \left[\frac{\mathrm{mol}}{\mathrm{cm}_p^3\ \mathrm{min}}\right] \tag{3.1}$$

where ϵ is the particle porosity, C_i is the species concentration, D_i is the species diffusion coefficient, ν_{il} is the stoichiometry of species i in leaching reaction l, and S_l is the rate term for the mineral leaching reaction l (discussed in Section 3.2.2). The last term on the RHS of Eq.(3.1) accounts for the microbial oxidation reactions at the particle surface. Here, ν_{im} is the stoichiometry coefficient species i in microbial oxidation reaction m, and S_{surf} is the microbial oxidation term at the particle surface (discussed in Section 3.2.2).

The boundary conditions for Eq.(3.1) are:

$$\frac{\partial C_i}{\partial r}(0,t) = 0 , \tag{3.2}$$

$$-D_i \frac{\partial C_i}{\partial r}\bigg|_{r=R} = -k_{\text{bs}}(C_{\text{b}i} - C_{Ri}) , \tag{3.3}$$

where $C_{\text{b}i}$ is the bulk concentration of the species, and k_{bs} is the mass transfer coefficient at the particle surface/bulk interface. At initial time, a uniform concentration of species i in size class j is assumed

Changes in the concentration of a chemical species in the bulk solution result from exchange at the particle surface/bulk interface, exchange at the bulk/flowing solution interface, and the rate of formation (or consumption) of the species within the bulk. This is represented by the mass balance equation below:

$$\varepsilon_{\text{b}} \frac{dC_{\text{b}i}}{dt} = -3\epsilon\varepsilon_{\text{p}} \sum_j \frac{w_j}{R_j} k_{\text{bs}}(C_{\text{b}i} - C_{R_j}) + k^*_{\text{bf}}(C_{\text{f}i} - C_{\text{b}i}) + \varepsilon_{\text{b}} \sum_m \nu_{im} S_{\text{bulk},m} \cdot \quad \left[\frac{\text{mol}}{\text{cm}^3 \text{ min}}\right] \tag{3.4}$$

The symbols are defined in the nomenclature. The first term on the RHS in Eq.(3.4) represents average rate of exchange between all particles and the bulk solution, (where the subscript j stands for particle size class). The second term represents the rate of exchange at the bulk/flowing solution interface. k^*_{bf} is the mass transfer rate at this interface, given by

$$k^*_{\text{bf}} = k_{\text{bf}} \cdot \frac{\text{exchange surface area}}{\text{agglomerate volume}} , \tag{3.5}$$

where k_{bf} is the mass transfer coefficient at the bulk/flowing solution interface. The third term represents the generation or consumption of species in the bulk as a result of microbial oxidation reactions. S_{bulk} is discussed in Section 3.2.2.

The rate of change of the concentration of a chemical species in the flowing phase is due to advection, and the rate of exchange with the bulk solution. This is given by the

equation below:

$$\varepsilon_{\mathrm{f}}\frac{\partial C_{\mathrm{f}i}}{\partial t} = -u\frac{\partial C_{\mathrm{f}i}}{\partial z} - k^{*}_{\mathrm{bf}}(C_{\mathrm{f}i} - C_{\mathrm{b}i}) \ , \qquad \left[\frac{\mathrm{mol}}{\mathrm{cm}^3\ \mathrm{min}}\right] \tag{3.6}$$

where u is the superficial velocity of the flowing solution.

Microbial Species

The overall population density of a given microbial species i, N_i (cells/$\mathrm{cm}^3_{\mathrm{agg}}$), consists of adsorbed microbes $N_{\mathrm{a},i}$ (cells/$\mathrm{g}_{\mathrm{ore}}$) and desorbed microbes $N_{\mathrm{d},i}$ (cells/$\mathrm{cm}^3_{\mathrm{bulk}}$) related by

$$N_i = \varepsilon_{\mathrm{p}}\rho(1-\epsilon)N_{\mathrm{a},i} + \varepsilon_{\mathrm{b}}N_{\mathrm{d},i} \ . \tag{3.7}$$

The rate of change of the adsorbed population for a species i and size class j is:

$$\left[\frac{dN_{\mathrm{a}}}{dt} = N_{\mathrm{a}}(\mu_{\mathrm{a}} - k_{\mathrm{death}})\right]_{ij} \ , \qquad \left[\frac{\mathrm{cells}}{\mathrm{g}_{\mathrm{ore}}\ \mathrm{min}}\right] \tag{3.8}$$

where μ_{a} is the growth rate of the adsorbed microbes and k_{death} is the specific death rate of the microbes. The forms of the microbial growth and death rates are discussed later in Section 3.2.2.

The rate of change of desorbed population is expressed as:

$$\varepsilon_{\mathrm{b}}\frac{dN_{\mathrm{d}}}{dt} = \varepsilon_{\mathrm{b}}N_{\mathrm{d}}(\mu_d - k_{\mathrm{death}}) + k^{*}_{\mathrm{bf}}(N_{\mathrm{f}} - N_{\mathrm{d}}) \ , \qquad \left[\frac{\mathrm{cells}}{\mathrm{cm}^3\ \mathrm{min}}\right] \tag{3.9}$$

where μ_{d} is the growth rate of the desorbed microbes, and N_{f} is the population density in the flowing phase.

The rate of change of the microbial population in the flowing solution is due to advection, and rate of exchange with the bulk solution. This is given by the equation below:

$$\varepsilon_{\mathrm{f}}\frac{\partial N_{\mathrm{f}}}{\partial t} = -u\frac{\partial N_{\mathrm{f}}}{\partial z} - k^{*}_{\mathrm{bf}}(N_{\mathrm{f}} - N_{\mathrm{d}}) \ . \qquad \left[\frac{\mathrm{cells}}{\mathrm{cm}^3\ \mathrm{min}}\right] \tag{3.10}$$

The adsorbed and desorbed microbial populations are coupled by the adsorption/desorption process. The adsorption of a species onto a solid offering sites for adsorption can be simply expressed as a reaction proceeds rapidly to equilibrium:

$$C_{\mathrm{ads}} \rightleftharpoons C_{\mathrm{des}} \ . \tag{3.11}$$

Adsorption kinetics can be suitably described by adsorption isotherms (Petersen, 1998). Single and multicomponent adsorption models were summarized by Volesky (2003). Three

well known adsorption isotherms are Freundlich, based on an empirical equation, Langmuir, based on an assumed mono-layer adsorption, and BET, based on multilayer adsorption considerations. The Langmuir isotherm model is used in this work.

The Langmuir isotherm model originates from gas adsorption on catalysts. The major assumptions of this model are:

1. Adsorption cannot proceed beyond monolayer coverage.
2. All surface sites are equivalent and can accommodate, at most, one adsorbed atom.
3. The ability of an atom to adsorb at a given site is independent of the occupation of neighboring sites.

At equilibrium, the rate of adsorption of the desorbed species equals the rate of desorption of adsorbed species. The Langmuir isotherm is then written as:

$$\frac{C_{\text{ads}}}{C_{\text{adsmax}}} = \frac{K_{\text{a}} C_{\text{des}}}{1 + K_{\text{a}} C_{\text{des}}} , \tag{3.12}$$

In Eq.(3.12), C_{ads} is the adsorbed concentration (mol/g), C_{adsmax} is the concentration corresponding to complete a mono-molecular layer (mol/g), C_{des} is the desorbed concentration (mol/L), and K_{a} is the adsorption equilibrium constant (L/mol).

Hence, the Langmuir isotherm for the microbial model is:

$$\frac{N_{\text{a}}}{N_{\text{a max}}} = \frac{K_{\text{a}} N_{\text{d}}}{1 + K_{\text{a}} N_{\text{d}}} . \tag{3.13}$$

3.2.2 Reaction Model

Sulfide Mineral Leaching

The rate of the sulfide mineral leaching reactions is a function of temperature, reactant concentration and the unreacted mineral fraction. It can be expressed in terms of the rate of change mineral conversion X, by rewriting the mineral grade g as $g_0 X$, where g_0 is the initial mineral grade (Dixon and Petersen, 2003). Hence,

$$S = \rho(1-\epsilon)\frac{\partial g}{\partial t} = \rho(1-\epsilon) g_0 \frac{\partial X}{\partial t} , \tag{3.14}$$

where

$$\frac{\partial X}{\partial t} = k(T) f(C) h(X) . \tag{3.15}$$

$k(T)$ is the temperature dependent rate constant given by Arrhenius law:

$$k(T) = k_{\text{ref}} \exp\left(-\frac{E}{R}\left[\frac{1}{T} - \frac{1}{T_{\text{ref}}}\right]\right) . \tag{3.16}$$

where k_{ref} is the Arrhenius constant at the reference temperature T_{ref}.

The concentration function $f(C)$ is given as:

$$f(C) = \frac{C_{Fe^{3+}}}{(k_A + C_{Fe^{3+}})^{1-m}(k_B + C_{Fe^{2+}})^m} . \tag{3.17}$$

Eq.(3.17) is from work by Dixon and Petersen (2003), and is better understood in the light of leaching chemistry.

Mineral leaching is an electrochemical process, involving the movement of charged species (ions and electrons) across the solid-liquid phase boundary (Crundwell, 1997). Both oxidative (for instance, copper sulfide mineral leaching by ferric ions) and non-oxidative (for instance, copper oxide mineral leaching by acid) dissolution of minerals are electrochemical in nature. The rate of an electrochemical reaction is dependent on the potential difference across the electrode-solution interface, the concentration of the reacting species, and temperature (Crundwell, 1997).

Dissolution reactions that result in the change of the oxidation state of the mineral can be separated into anodic and cathodic half-reactions. For example the dissolution of a mineral M in an aqueous solution containing the oxidant B^{3+}, and represented by the reaction:

$$M(s) + B^{3+}(aq) \rightarrow M^{+}(s) + B^{2+}(aq) \tag{3.18}$$

can be split into the anodic dissolution of the metal M,

$$M(s) \rightarrow M^{+}(s) + e^{-}(s), \tag{3.19}$$

and the cathodic reduction of the oxidant B^{3+}

$$B^{3+}(aq) + e^{-}(s) \rightarrow B^{2+}(aq). \tag{3.20}$$

The rate of dissolution of M is then given by (Crundwell, 1997; Nicol and Needes, 1975):

$$r_{\text{diss}} = k_a \left(\frac{k_c[B^{3+}]}{k_a + k_c^1[B^{2+}]}\right)^{\alpha} . \tag{3.21}$$

where k_a is the rate constant for the anodic half reaction, k_c and k_c^1 are the rate constants for the cathodic half reaction of $[B^{3+}]$ and $[B^{2+}]$ respectively, and α is the charge-transfer coefficient.

The terms in Eq.(3.21) are rearranged to a form similar to Eq.(3.17) as shown below:

$$r_{\text{diss}} = \left(\frac{k_a k_c^{\alpha}}{k_{c1}^{\alpha}}\right) \frac{[\mathrm{B}^{3+}]}{([\mathrm{B}^{3+}])^{1-\alpha}(\frac{k_a}{k_{c1}} + [\mathrm{B}^{2+}])^{\alpha}} , \tag{3.22}$$

Comparing Eq.(3.17) to Eq.(3.22),

$$\begin{aligned} m = \alpha; \quad & C_{Fe^{3+}} = [\mathrm{B}^{3+}]; \quad C_{Fe^{2+}} = [\mathrm{B}^{2+}] \\ k_A = 0; \quad & k_B = \frac{k_a}{k_{c1}}; \qquad K_C = \frac{k_a k_c^{\alpha}}{k_{c1}^{\alpha}} \quad ; \end{aligned} \tag{3.23}$$

assuming K_C is a rate constant that is factored out. Eq.(3.17) can represent any one of the following cases

1. Anodic mineral decomposition is rate controlling: $k_a \ll k_{c1}[\mathrm{B}^{2+}]$, in which case $k_B \ll C_{\mathrm{Fe}^{2+}}$, and $k_A = 0$. $f(C)$ is then a function of the ferric to ferrous ratio:

$$f(C) \approx \left(\frac{C_{Fe^{3+}}}{C_{Fe^{2+}}}\right)^m . \tag{3.24}$$

2. Cathodic ferric reduction is rate controlling: $k_a \gg k_{c1}[\mathrm{B}^{2+}]$, in which case $k_B \gg C_{\mathrm{Fe}^{2+}}$, and $k_A = 0$. $f(C)$ depends on the ferric concentration alone:

$$f(C) \approx (C_{Fe^{3+}})^m , \tag{3.25}$$

where K_C is now slightly different. $K_C = k_a \left(\frac{k_c}{k_a}\right)^m$ (Compare with K_C in Eq.(3.23)).

3. A third case represented by Eq.(3.17) is when ferric mass transfer is rate controlling, in which case $k_A \neq 0$. Here $k_A \gg C_{\mathrm{Fe}^{3+}}$, $k_B \gg C_{\mathrm{Fe}^{2+}}$, and $f(C)$ is of the form:

$$f(C) \approx \frac{C_{\mathrm{Fe}^{3+}}}{k_A^{(1-m)} k_B^m} . \tag{3.26}$$

$h(X)$ is a topological term which accounts for the change in reacting surface. A power law, proposed by Dixon and Hendrix (1993), is implemented:

$$h(X) = (1 - X)^{\phi} . \tag{3.27}$$

ϕ is an empirical parameter that usually falls between 0.5 and 2.0 (Petersen and Dixon, 2002a).

Biological Phenomena

The models for microbial oxidation, growth, death and temperature dependence are presented in this section. The effect of sulfur oxidizing microbes on mineral leaching will be investigated in Chapter 4. Hence in addition to the mathematical model for iron oxidizers, the mathematical model for sulfur oxidizers is also outlined in this section.

Microbial Oxidation

The rate of microbial oxidation is related to the microbial growth rate through a stoichiometric factor, the yield coefficient Y, and through the cell maintenance rate, k_m (Dixon and Petersen, 2003; Ojumu et al., 2005). The yield coefficient is the number of cells formed per mol of substrate oxidized. The maintenance rate is the moles of substrate used for cell maintenance per cell per unit time. The microbial maintenance rate is modified by temperature. Hence, the rate of microbial oxidation is:

$$S = N_k \left(\frac{\mu_k}{Y} + k_m f(T) \right) . \tag{3.28}$$

The subscript k identifies either the adsorbed or planktonic microbial population.

Microbial Growth Rate

The development of rate equations for iron and sulfur oxidizing bacteria in heap bioleaching is a dynamic field of study. Over the years, several rate equations have been formulated, some of which are compared in Ojumu et al. (2005). It is worthy of note that most of these rate equations appear to be of a modified Monod form. In the absence of an established rate equation, the formulation of Dixon and Petersen (2003) is adopted in the present model.

Microbial growth rate is expressed as a maximum microbial growth modified by limiting factors:

$$\mu_i = \mu_{i,\max} F(T, C_{O_2}, C_{\text{acid}}, N, C_{Fe^{2+}} \text{ or } C_S, \ldots) . \tag{3.29}$$

The growth rate of iron oxidizing microbes is given by:

$$\mu_{Fe} = \mu_{Fe,\max} \cdot f(T) \cdot \left(1 - \exp\left[-\frac{C_{\text{acid}}}{C_{\text{acidlim}}} \right] \right) \cdot \left(\frac{C_{Fe^{2+}}}{K_{Fe^{2+}} + C_{Fe^{2+}}} \right) \cdot \left(\frac{C_{O_2}}{K_{O_2} + C_{O_2}} \right) \cdot \left(\frac{K_I}{K_I + N_{Fe}} \right) \tag{3.30}$$

Eq.(3.30) shows a dependence of the growth rate of iron oxidizing microbes on temperature, pH, ferrous ion concentration, dissolved oxygen concentration, and cell crowding. The cell crowding factor becomes inhibitory as the microbial population increases beyond the cell inhibition constant K_{I}. The substrate concentrations in Eq.(3.30) are either particle surface or bulk concentrations, depending on whether adsorbed or desorbed microbial populations, respectively, are under consideration.

A similar expression is written for the growth rate of sulfur oxidizing microbes. Sulfur oxidizing microbes grow by oxidizing elemental sulfur at the surface of the ore particles. Therefore, it will be assumed that planktonic sulfur oxidizers do not grow. The growth rate of adsorbed sulfur oxidizers is given by:

$$\mu_{\mathrm{S}} = \mu_{\mathrm{S,max}} f(T) \cdot \left(\frac{g_{\mathrm{S}}}{K_{\mathrm{S}} + g_S}\right) \cdot \left(\frac{C_{\mathrm{O_2}}}{K_{\mathrm{O_2}} + C_{\mathrm{O_2}}}\right) \cdot \left(\frac{K_{\mathrm{I}}}{K_{\mathrm{I}} + N_{\mathrm{S}}}\right) \tag{3.31}$$

where g_S is the sulfur grade.

Microbial Death Rate

A constant specific natural death rate is implemented for k_{death}. k_{death} is different for each microbial species. Other models for the death rate are possible. For example, Dixon and Petersen (2003) implement a combination of an endogenous decay rate and a temperature dependent death rate.

Microbial Temperature Function

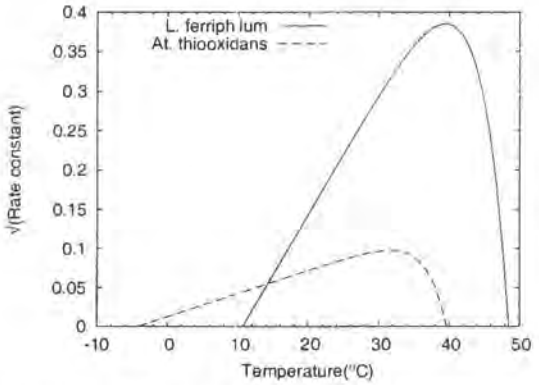
The dependence of microbial growth rate on temperature can be described by the Ratkowsky equation (Ratkowsky et al., 1983). Defining $f(T) = F(T)/F(T_{\mathrm{opt}})$ as the normalized temperature function, the Ratkowsky function $F(T)$ is given by:

$$F(T) = \sqrt{r} = b(T - T_{\mathrm{min}})(1 - \exp(c(T - T_{\mathrm{max}}))) \,, \tag{3.32}$$

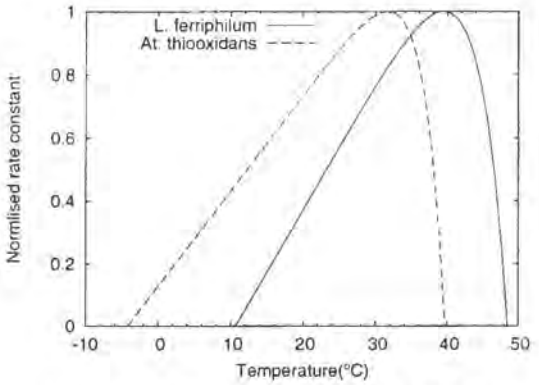
where r is the growth rate constant, T is the temperature (in Kelvin), and b, c are fitting parameters. $T_{\mathrm{min}}, T_{\mathrm{max}}$ are the theoretical extrapolated minimum and maximum temperatures for microbial growth, while T_{opt} is the temperature at which optimal microbial growth rate is achieved.

The Ratkowsky equation is a purely empirical equation that was developed to fit experimental data. Zwietering et al. (1991) evaluated the suitability and usefulness of

different models that describe the relationship between microbial growth and temperature by statistical analysis of large amounts of experimental data, and concluded that modified forms of the Ratkowsky equation were the most suitable. Franzmann et al. (2005) have reported the Ratkowsky equation parameter values ($T_{\min}, T_{\max}, T_{\text{opt}}, b, c$) for selected common mineral leaching microbes growing on either ferrous ion or sulfur compounds. They expressed the rate constant r in Eq.(3.32) as the inverse of a generation time or the time taken to reach a specific condition (e.g. the time it takes a culture to increase its optical density to 0.3 absorbance units).



(a) The rate constant for *L. ferriphilum* is the time taken to double the Ferric concentration while the rate constant for *At. thiooxidans* is the time to increase the SO_4-S concentration to 3000mg/L.



(b) Graphs in (a) normalized by $F(T_{\text{opt}})$

Figure 3.3: Ratkowsky plots for *L. ferriphilum* and *At. thiooxidans*.

Temperature parameters for *L. ferriphilum* and *At. thiooxidans* are chosen in this work for iron and sulfur oxidizers respectively. Based on parameters reported by Franzmann et al. (2005), the Ratkowsky plots for these microbes are shown in Fig.3.3. *L. ferriphilum* exists between the temperature range 10.7–48.5 °C and has $T_{\text{opt}} = 38.6$ °C, while *At. thiooxidans* exists between the temperature range -4.2–39.7 °C and has $T_{\text{opt}} = 32.8$ °C.

3.3 Conclusion

An agglomerate scale heap bioleaching model has been proposed. This model includes transport, chemical and microbial processes occurring at the level of a cluster of different sized ore particles. The model was developed in the context of a representative unit volume of a heap, comprising of a solid volume fraction (ore particles), and a liquid volume fraction (stagnant and flowing liquid phases).

Particle scale kinetics was modelled by grouping particles into size classes, and solving a diffusion-reaction equation for each size class. A well mixed assumption was made for the stagnant phase, which is reasonable, comparing the scale of the agglomerate to the scale of the heap. The model developed here did not explicitly cater for the gas phase (flowing gas and gas pockets). Instead, a certain concentration of dissolved oxygen was specified in the incoming liquid phase.

Two significant assumptions made in the model are isothermal conditions, and constant incoming dissolved oxygen concentration. The mineral leaching reactions (especially covellite leaching), and the microbial growth rate, are temperature dependent. Also, the rate of gaseous oxygen uptake into solution, which is a temperature dependent mass transfer step, is an important parameter in heap modelling, since oxygen is a key reactant in the microbial oxidation reactions. Hence, the present model can be made more realistic by incorporating these processes.

CHAPTER 4

Numerical Results

In this chapter, the agglomerate model is applied to study copper bioleaching from an ore that has chalcocite and pyrite as its major components. The response of the cumulative copper recovery to various model parameters is investigated.

4.1 Description of Case Study

The agglomerate model is tested on a theoretical case study ore that contains a mixture of chalcocite and pyrite, in the presence of iron and sulfur oxidizing microbes.

Chalcocite leaches by a two-stage mechanism. In the first stage, chalcocite reacts with ferric ions to form a covellite-like intermediate mineral (sometimes referred to as *blaubleibender* (Dixon and Petersen, 2003)), and releases about 40% of the total copper (Eq.(4.1)). In the second stage, covellite formed in the first stage reacts with ferric ions to release the remaining 60% of the total copper, and elemental sulphur (Eq.(4.2)).

$$Cu_2S + 1.6Fe^{3+} \rightarrow 0.8Cu^{2+} + 1.6Fe^{2+} + Cu_{1.2}S \tag{4.1}$$

$$Cu_{1.2}S + 2.4Fe^{3+} \rightarrow 1.2Cu^{2+} + 2.4Fe^{2+} + S^0 \tag{4.2}$$

The intrinsic rate of the first stage is high even at room temperature, and is limited primarily by the diffusion of ferric ions to the reaction sites in the ore particle. This stage has a relatively low activation energy, (values from 4–25 kJ/mol have been reported (Petersen and Dixon, 2003)). The intrinsic rate of the second stage is much slower, and it has a much higher activation energy that the first stage (values from 55–105 kJ/mol have been reported); hence the leaching rate increases significantly with temperature (Petersen and Dixon, 2003). The second stage shows a near half-order dependence on the ferric to ferrous ratio, which suggests control by charge transfer in the anodic half-reaction (Petersen and Dixon, 2003).

Pyrite is often found with chalcocite in practice. It leaches by reaction with ferric ions to release ferrous ions, sulfates, and (depending on the redox potential), elemental

sulphur:

$$\begin{aligned} FeS_2 + (14 - 6\beta)Fe^{3+} + 4(2-\beta)H_2O \rightarrow \\ (15 - 6\beta)Fe^{2+} + 8(2-\beta)H^+ + (2-\beta)SO_4^{2-} + \beta S^0 \end{aligned} \tag{4.3}$$

β in Eq.(4.3) is the fraction of pyrite that goes to elemental sulphur rather than sulphate. It takes a value between 0 and 1 depending on the redox potential (Dixon and Petersen, 2003). In this study, it is assumed that $\beta = 0$, that is, no elemental sulfur is formed from pyrite.

In addition to Eq.(4.1) – Eq.(4.3) above, acid consumption by gangue is also considered, according to the reaction below:

$$H_2SO_4 + MO \rightarrow MSO_4 + H_2O \tag{4.4}$$

where MO is the gangue mineral.

The microbial oxidation reactions (ferrous and sulfur oxidation) are given by the equations below:

$$Fe^{2+} + 0.25O_2 + 0.5H_2SO_4 \xrightarrow{\text{microbes}} Fe^{3+} + 0.5H_2O \tag{4.5}$$

$$S^0 + 1.5O_2 + H_2O \xrightarrow{\text{microbes}} H_2SO_4 \tag{4.6}$$

From Eq.(4.1)-Eq.(4.4), the following stoichiometry matrix for the leaching reactions in the ore particles is formed (refer to Eq.(3.1)):

$$\nu_{il} = \begin{array}{c} \begin{matrix} \text{Eq.(4.1)} & \text{Eq.(4.2)} & \text{Eq.(4.3)} & \text{Eq.(4.4)} \end{matrix} \\ \begin{pmatrix} 0.8 & 6.0 & 0.0 & 0.0 \\ -1.6 & -2.4 & 14-6\beta & 0.0 \\ 1.6 & 2.4 & 15-6\beta & 0.0 \\ 0.0 & 0.0 & 8-4\beta & -1.0 \end{pmatrix} \end{array} \begin{matrix} Cu^{2+} \\ Fe^{3+} \\ Fe^{2+} \\ \text{Acid} \end{matrix} , \tag{4.7}$$

where the rows of the matrix ν_{il} (denoted by subscript i) represent the dissolved species, and the columns (denoted by subscript l) represent the mineral leaching reactions.

Simulations were carried out on three particle size classes of radii 1mm, 3mm and 5mm. The following initial conditions were used:

$$C_{f,Fe^{2+}}(z,0) = 0.1 \text{ g/L}, \quad C_{f,Fe^{3+}}(z,0) = 0.1 \text{ g/L}, \quad C_{f,O_2}(z,0) = 0.008 \text{ g/L} \tag{4.8}$$

$$C_{f,H_2SO_4}(z,0) = 8 \text{ g/L}, \quad N_f(z,0) = 10^{10} \text{ cells/L}, \quad C_{p,i}(r,0) = 0 \tag{4.9}$$

$$C_{b,i}(0) = C_{f,i}(z,0), \quad N_a(0) = N_f(z,0) \text{ cells/L}, \quad N_a(0) = 0 \text{ cells/g} \tag{4.10}$$

i in Eq.(4.9) and Eq.(4.10) is one of the dissolved species [Cu, Fe^{3+}, Fe^{2+}, O_2, H_2SO_4]. Initial copper concentration in the system was set to zero. The initial bulk concentration of all chemical species was set equal to the initial concentration in the flowing phase, as shown in Eq.(4.10).

Other baseline simulation parameters are listed in Table 4.1–Table 4.4. The parameter values for this case study are taken from literature, and modelling and experimental work done by Petersen and Dixon (2003). The solid volume fraction in the agglomerate is 0.6, the stagnant liquid volume fraction is 0.3, and the flowing liquid volume fraction is 0.1. A baseline temperature of 38.6°C is chosen, to correspond with the optimum operating temperature of the iron oxidizing microbes. The proportion (by mass) of pyrite to chalcocite in the ore is roughly 5:1, which represents the proportion that was found in a Chilean chalcocite ore sample (Dixon and Petersen, 2003).

Parameter	Unit	Fe Ox (Meso)	S Ox (Meso)
$N_{f,inlet}$	cells/L	10^{10}	10^9
M (molar mass)	g/mol	10^{12}	10^{12}
Y	cells/mol substrate	2×10^{12}	2×10^{11}
μ_{max}	1/min	0.0017	0.00083
k_{death}	1/min	0.00017	0.00001
k_m	mol substrate/cell/min	0.0	0.0
N_{amax}	cell/g	1.5×10^9	1.5×10^9
K_a	L/cell	67×10^{-12}	17×10^{-12}
$K_{substrate}$	mol/L (K_{Fe}) or mol/g (K_S)	K_{Fe}= 0.0001	K_S = 0.00006
K_{acid}	mol/L	0.01	–
K_{O_2}	mol/L	0.00005	0.00005
K_I	cells/L	10^{12}	10^{12}
T_{min}	°C	9.6	-4.2
T_{max}	°C	49.5	39.7
T_{opt}	°C	38.6	32.8
b	(Ratkowsky fitting parameter)	0.01551	0.00293
c	(Ratkowsky fitting parameter)	0.22061	0.30891

Fe Ox (Meso) = Iron oxidizing mesophiles, S Ox (Meso) = Sulfur oxidizing mesophiles.

Table 4.1: Microbial Parameters

Parameter	Unit	Copper	Ferric	Ferrous	Acid	Oxygen
D	m^2/s	10^{-9}	10^{-9}	10^{-9}	3×10^{-9}	10^{-9}
$C_{f,inlet}$	g/L	0.0	0.1	0.1	8	0.0008
M	g/mol	63.54	55.85	55.85	98.07	32
k_{ref}	1/min	–	–	31.17	9.3×10^{-4}	–
T_{ref}	°C	–	–	100	20	–
E	kJ/mol	–	–	68.6	20	–

Table 4.2: Species Parameters

Parameter	Unit	Chalcocite	Covellite	Pyrite
φ	–	1.3	0.6	2.0
m	–	0.124	0.5	0.5
k_{ref}	1/min	0.446	0.01	0.00005
T_{ref}	°C	35	75	55
E	kJ/mol	23.9	97.9	74.3
k_A	mol/L	0.154	0.0147	0.00001
k_B	mol/L	0.00001	0.00001	0.00001
ϕ	mol/kg_{ore}	0.0566	0.0566	0.3
M	g/mol	159.14	541.54	119.97

ϕ and m are non-dimensional.

Table 4.3: Mineral Parameters

Parameter	Unit	Value
ρ	g/cm^3	1.45
ϵ	cm^3/cm^3	0.08
ε_b	cm^3/cm^3	0.3
ε_f	cm^3/cm^3	0.1
T	°C	38.6
L	cm	20
u	cm/min	0.05
k_{sb}	cm/min	0.1
k^*_{bf}	1/min	0.1

L = Length of side of unit volume.

Table 4.4: User Specified Parameters

A discussion of the numerical methods used in this work is given in Appendix A. The consistency of the simulation results was checked by performing a mass balance at the end of each simulation. Total copper formed, based on the final conversion of the copper minerals, was compared to the total dissolved copper obtained during the simulation (that is, the sum of the amount of copper in the agglomerate at the end to the simulation, and the total copper that has been carried out of the system by the flowing phase). The maximum relative error obtained from this calculation, for all the simulations that were run in this thesis, was <0.7%. More discussion on the error is given in Appendix A.

4.2 Interpretation of Base Case Simulation Results

The bulk concentration of copper, and the average conversion of chalcocite, covellite and pyrite in each of the particle size classes, for leaching in the presence of iron oxidizing microbes, are shown in Fig.4.1. Each size class is equally weighted. The average conversion graphs for the size classes (see Fig.4.1(b)-(d)), were obtained by taking a volume average of mineral conversion profiles:

$$\overline{X}_{ij} = \frac{4\pi}{V_j}\int_0^{R_j} X_{ij}(r) r_j^2 \, dr_j \,, \tag{4.11}$$

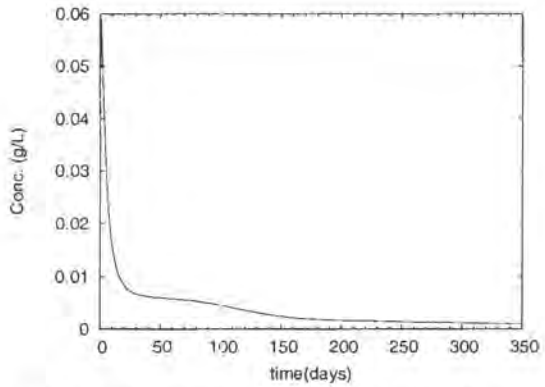
$$= \frac{3}{R_j^3}\int_0^{R_j} X_{ij}(r) r_j^2 \, dr_j \,, \tag{4.12}$$

$$\text{where } V_j = \frac{4}{3}\pi R_j^3 \,, \tag{4.13}$$

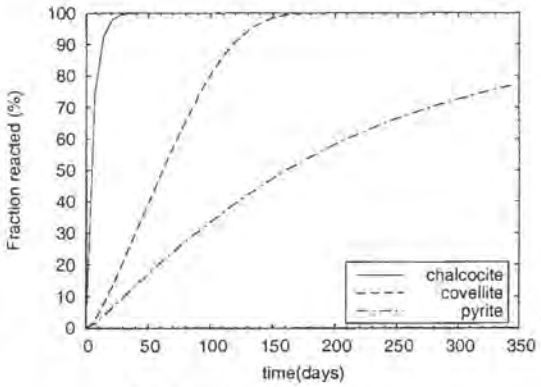
for mineral i and size class j.

The first ten days of the simulation are dominated by chalcocite leaching. Fig.4.1(b)-(d) show rapid conversion of chalcocite in this time period, with about 90% chalcocite conversion in the 1mm radius size class. The bulk concentration of copper jumps to 0.06 g/L within the first few hours of the simulation (see Fig.4.1(a)), due to rapid introduction of copper into the system by chalcocite leaching (especially in the 1mm radius size class). However, the bulk copper concentration falls to less than 0.01 g/L in the first ten days of simulation, as the flowing phase carries copper away from the agglomerate.

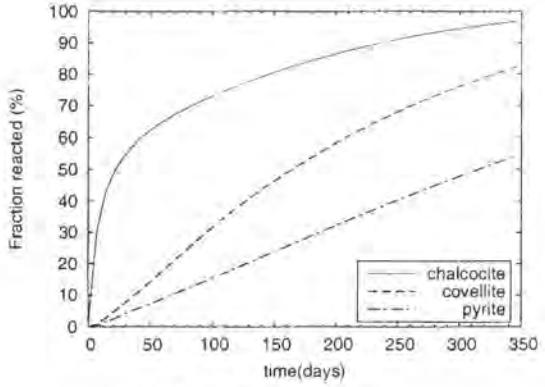
The rate of decrease in the bulk concentration of copper slows down considerably from day 20. A gentle concavity in the bulk copper concentration is observed between days 20 and 150. This is mostly due to covellite conversion in the 1mm size class. Comparing Fig.4.1(a) to Fig.4.1(b), it is seen that the concavity starts at about the completion of



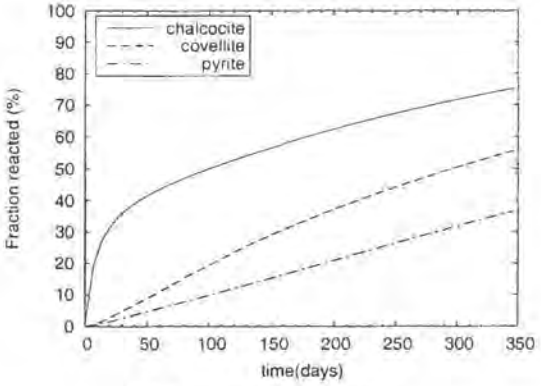
(a) Copper concentration in stagnant phase.



(b) 1mm radius size class



(c) 3mm radius size class

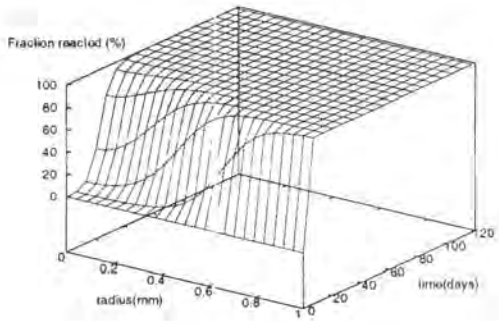


(d) 5mm radius size class

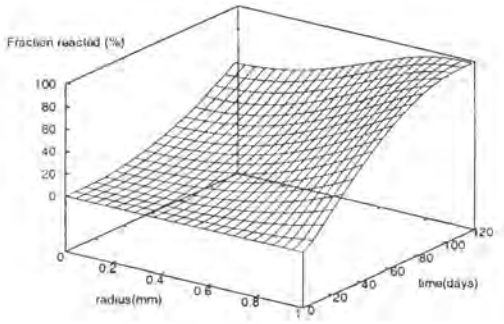
Figure 4.1: Bulk concentration of copper and average mineral conversion profiles for baseline parameters.

chalcocite leaching in the 1mm size class, and ends at about the completion of covellite leaching in this size class. As stated earlier, covellite leaching releases 60% of the total copper. Therefore, covellite conversion in the 1mm size class between days 20 and 150 introduces a significant amount of copper into the system, causing the observed concavity in the bulk copper concentration. From day 150, the bulk copper concentration is low and slowly declining, signalling the effect, on copper extraction, of diffusion limitation in the coarse ore particles.

Experimental evidence indicates that first stage chalcocite leaching is limited by the availability of reagents, while second stage chalcocite leaching is limited by mineral kinetics (Petersen and Dixon, 2003). This can be seen in Fig.4.2, which shows the progression of mineral leaching in the 1mm size class in more detail. Steep conversion profiles, typical

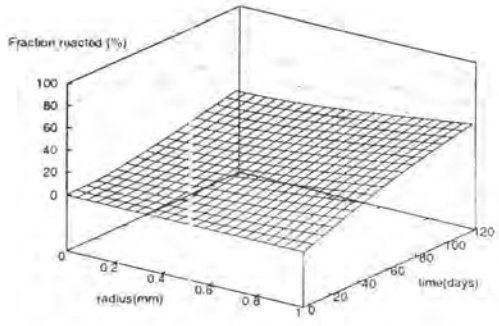


(a) Chalcocite conversion



(b) Covellite conversion

Figure 4.2: Mineral conversion profiles in the 1mm radius size class after 120 days. On the radial scale, 0 represents the center of the particle, and 1 the particle surface.



(c) Pyrite conversion

Figure 4.2: Contd.

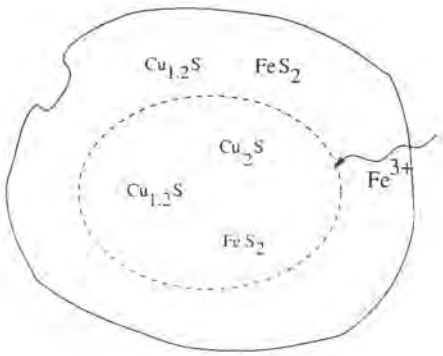
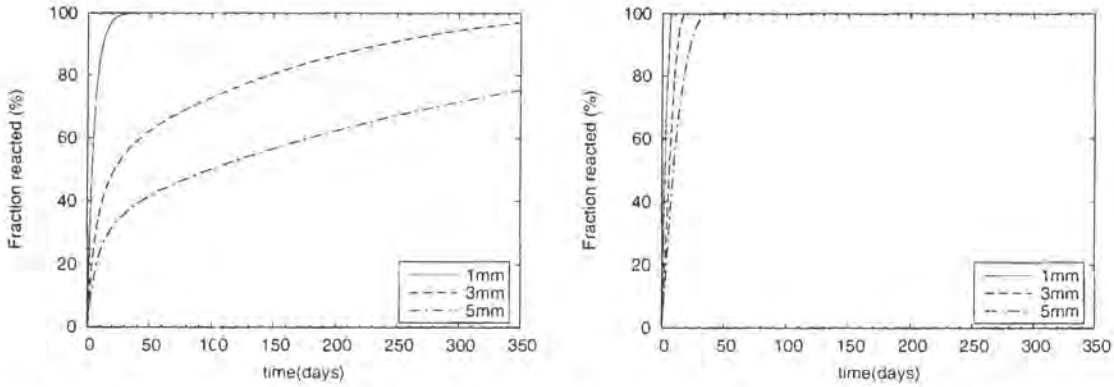


Figure 4.3: Mineral leaching in a coarse ore particle.

of fast chemical k netics relative to the diffusion rate of reagents into the particle, are observed for chalcocite, while covellite reacts more homogeneously. The successive nature of the first and second stages of chalcocite leaching is also apparent from Fig.4.2(a) and (b); covellite conversion is only in regions where chalcocite conversion is near completion. It is noted here that in the model implementation, it was assumed that all the covellite available from chalcocite was present at initial time. It is clear from Fig.4.2 that this assumption yields expected qualitative results, with respect to covellite conversion proceeding only in regions in the ore particles where chalcocite conversion has advanced considerably. Pyrite reacts more slowly than the copper minerals (as shown in Fig.4.2(c)).

The effect of diffusion distance on mineral leaching at the particle scale is illustrated

by Fig.4.1(d). Within the first 50 days of simulation, the minerals exhibit very different conversion rates, and the higher intrinsic rate of chalcocite reaction is apparent. Thereafter, the mineral conversion rates are quite similar, with the conversion rate of chalcocite slowing down considerably. This behavior is observed because as chalcocite leaching progresses into the coarse particles, ferric ions diffusing into the particle come in contact (and react with) the unreacted covellite and pyrite before they can to get to chalcocite deeper in the particle (Fig.4.3). Therefore, although chalcocite has a higher intrinsic reaction rate, it does not leach much faster than pyrite or covellite in the coarse ore particles. In the absence of covellite and pyrite, chalcocite conversion conversion in the three size classes in completed within the first 50 days of simulation, as shown in Fig.4.4(b). In finer particles, ferric ions have a shorter diffusion distance to travel, and so all the chalcocite is converted before this competitive leaching of minerals is established. Hence, the simulation results suggest that in coarse particles, covellite and pyrite in the unreacted region retard the rapid conversion of chalcocite observed in the finer particles.



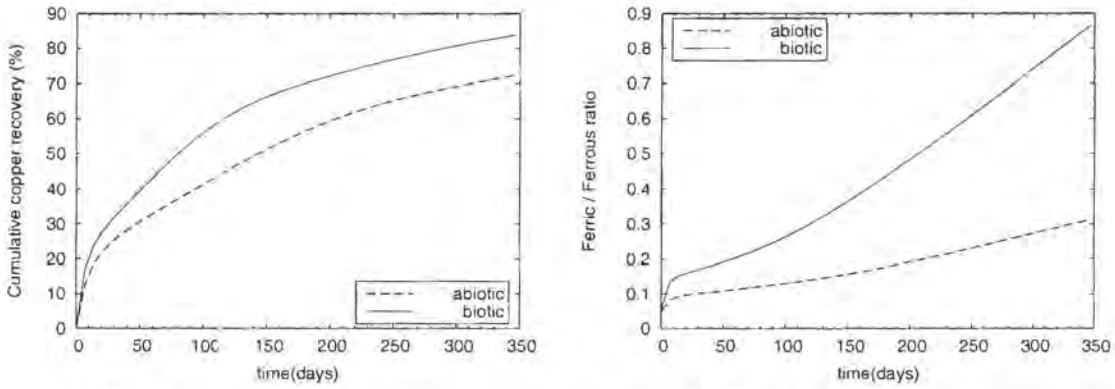
(a) Chalcocite conversion profiles in the presence of covellite and pyrite

(b) Chalcocite conversion profiles in the absence of covellite and pyrite

Figure 4.4: Chalcocite conversion profiles in the presence and absence of covellite and pyrite.

4.3 Comparison of Abiotic and Biotic Leaching Simulations

Fig.4.5 shows the copper recovery, and bulk ferric/ferrous ratio, for abiotic and biotic leaching simulations. Apart from the absence of microbes in the abiotic simulation, all other simulation parameters in the abiotic and biotic simulations are equal.



(a) Copper recovery in flowing phase (b) Bulk ferric/ferrous ratio

Figure 4.5: Comparison of abiotic and biotic leaching simulation results.

The action of iron oxidizing microbes in heap bioleaching is to increase ferric concentration by oxidizing ferrous according to Eq.(4.5), leading to more copper recovery. This is illustrated in Fig.4.5, which shows a higher bulk ferric/ferrous ratio, and higher copper recovery, in the biotic simulation compared with the abiotic simulation. Although the ferric/ferrous ratio for the biotic case is smaller than the values usually obtained in laboratory experiments (typically > 10), it shows that the model predicts expected behavior. A point to note is that the base case is operating under oxygen limitation, compared to typical laboratory conditions. As will be shown in Section 4.4.4, increasing the inlet oxygen concentration significantly increases the ferric/ferrous ratio (due to increased microbial activity), resulting in increased copper recovery.

4.4 Model Sensitivity

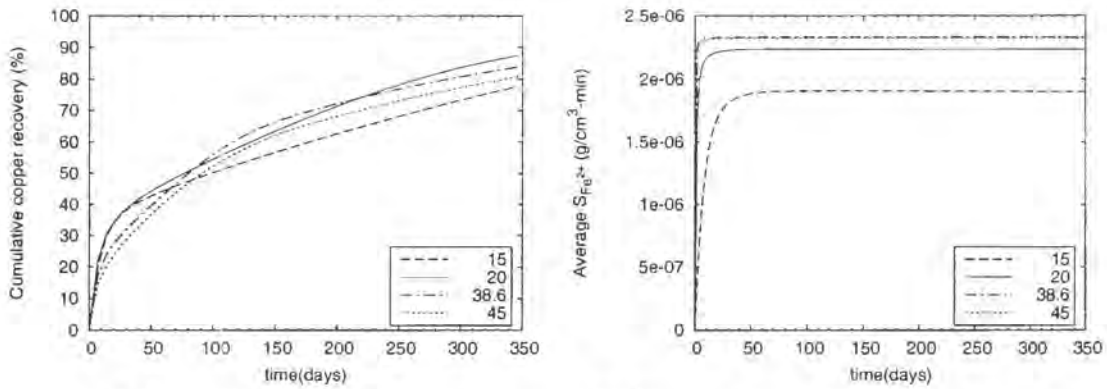
In this section, the effect of selected simulation parameters on the cumulative copper recovery is investigated. The parameters investigated are temperature, particle size distribution, solution flow rate, inlet substrate concentration, mass transfer parameters, Arrhenius rate constant, inlet microbial population, maximum microbial growth rate, and presence of sulfur oxidizers.

4.4.1 Effect of Temperature

Fig.4.6(a) shows the cumulative copper recovery for simulations run at 15°C, 20°C, 38.6°C, and 45°C, keeping all other baseline parameters constant. The baseline temperature is 38.6°C, chosen because it is the optimum operating temperature of *L. ferriphilum*, the iron oxidizing microbial species used in this work (see Franzmann et al., 2005). Chalcocite heap leaching usually operates in the neighborhood of 15–25°C, hence the choice of 15°C and 20°C as simulation temperatures. The average microbial ferrous oxidation rate, shown in Fig.4.6(b), was calculated as follows:

$$\overline{S}_{Fe^{2+}} = \varepsilon_p \sum_j w_j S_{Fe^{2+},j} + \varepsilon_b S_{Fe^{2+},bulk}, \tag{4.14}$$

where j is the size class index, and $S_{Fe^{2+}}$ is the microbial ferrous oxidation rate.



(a) Cumulative copper recovery (b) Average microbial ferrous oxidation rate

Figure 4.6: Effect of temperature on copper recovery and microbial ferrous oxidation rate.

As seen in Fig.4.6(a), at a temperature of 15°C or 45°C, copper recovery is lower than the baseline recovery after 350 days. However, at a temperature of 20°C, copper recovery is

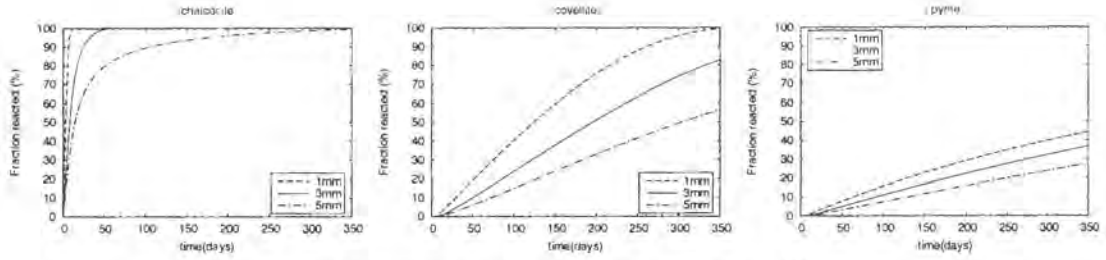
higher than the baseline recovery after 350 days, despite the fact that covellite and pyrite reaction rates are considerably lower at this temperature, and that microbial ferrous oxidation rate is lower than optimum at this temperature, as shown in Fig.4.6(b). The higher copper recovery at this temperature is maintained even after 700 days of simulation. This can be explained by considering the effect of temperature on the intrinsic reaction rate of the minerals, as follows.

As stated in Section 4.1, the activation energy of the first stage chalcocite leaching reaction is much lower than that of the second stage (covellite) leaching reaction. (The values E_{Cu_2S}=23.9 kJ/mol and $E_{Cu_{1.2}S}$=97.9 kJ/mol were used in this simulation). Hence, the leaching rate of covellite increases significantly with temperature, compared to chalcocite. The activation energy of pyrite is also much higher than that of chalcocite (E_{FeS_2}=74.3 kJ/mol was used), hence pyrite leaching is also more sensitive to temperature than chalcocite. This is illustrated in Fig.4.7, which shows the average conversion profiles of chalcocite, covellite and pyrite, in the different particle size classes, at 20°C, 38.6°C and 45°C.

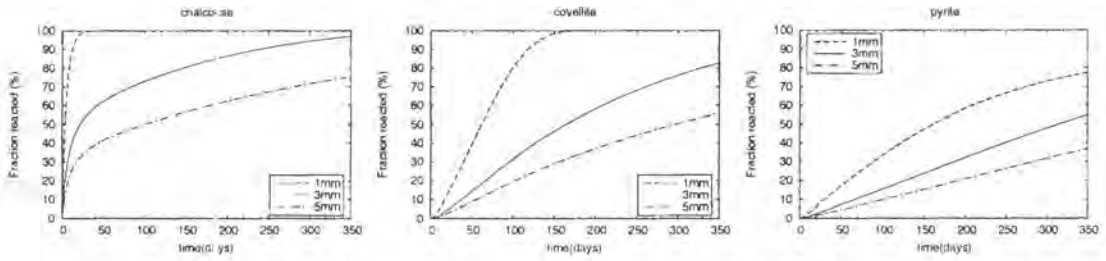
Covellite and pyrite reaction rates are lower at 20°C compared to 38.6°C. This results in the availability of more ferric ions for chalcocite leaching at this lower temperature. Consequently, chalcocite leaches to completion much earlier at 20°C compared to 38.6°C. On the other hand the early completion of chalcocite leaching implies that ferric ions are able to penetrate deep into the particle, resulting in an overall appreciable rate of covellite leaching, despite the lower temperature. This is clearly shown in Fig.4.8. Operating at a much lower temperature (15°C for instance), reduces the rate of covellite leaching even further, so that although the ferric ions are able to penetrate deep into the particle, the total covellite leached is not comparable to the amount leached at 38.6°C. Hence the overall copper recovery is much lower at 15°C (see Fig.4.6(a)).

At 45°C, the average microbial ferrous oxidation rate is indistinguishable from the rate at the optimum temperature of 38.6°C. This is because the normalized Ratkowsky function is still high at this temperature (see Fig.3.3(b)). Nevertheless, the cumulative copper recovery at 45°C is clearly lower than at 38.6°C (see Fig.4.6(a)). At 45°C, the rate of covellite and pyrite reaction is higher than at 38.6°C. This higher reaction rate leads to more competition for ferric ions among the minerals, and slows down chalcocite conversion (compare Fig.4.7(b) and (c)). The overall effect is that copper recovery is reduced, as shown in Fig.4.6(a).

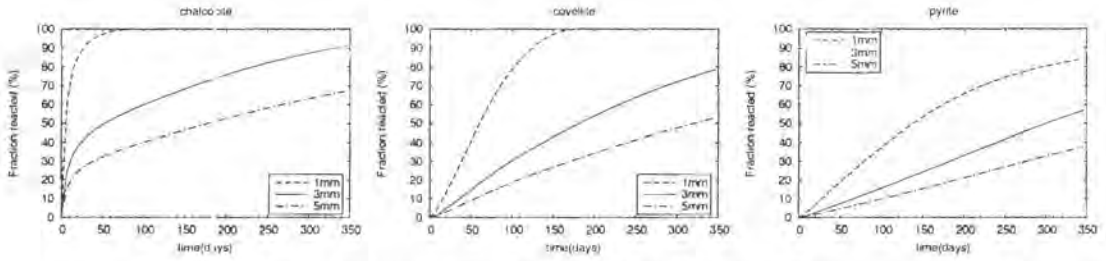
Competitive leaching of chalcocite, covellite and pyrite has been observed as a result



(a) Average mineral conversion profiles at 20°C



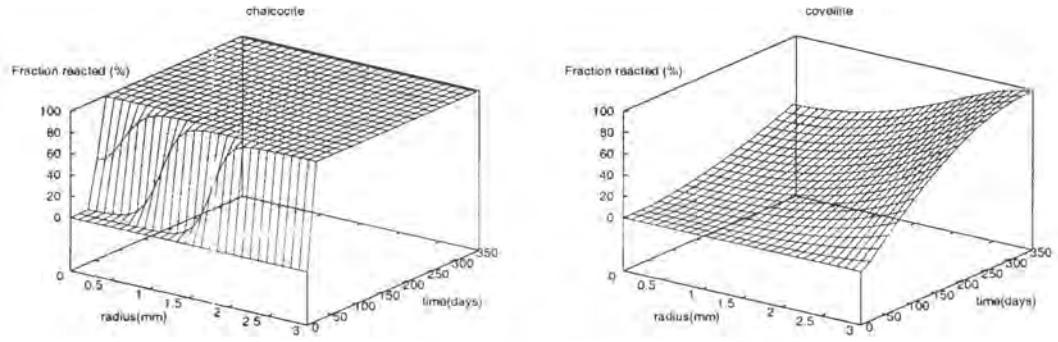
(b) Average mineral conversion profiles at 38.6°C



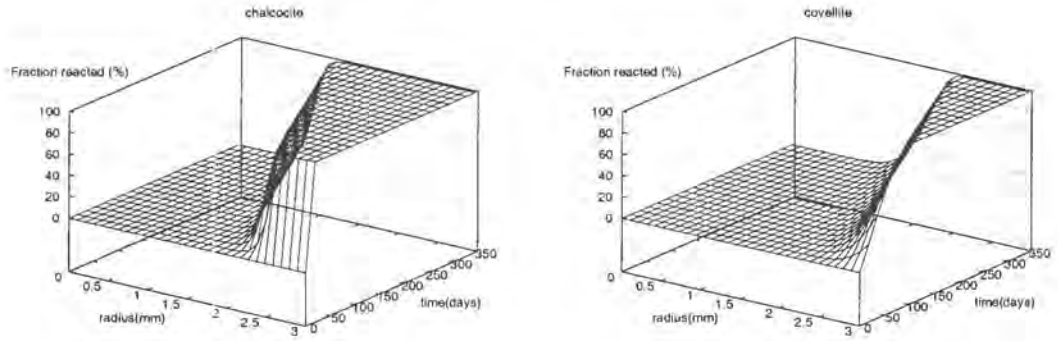
(c) Average mineral conversion profiles at 45°C

Figure 4.7: Average mineral conversion profiles at 20°C, 38.6°C and 45°C.

of the agglomerate temperature. By running the leaching simulation at a temperature that is lower than the optimal operating temperature of the microbes, more overall copper conversion is attained (despite the lower activity of the microbes and lower reaction rates), because the competition for ferric ions between the minerals is less. This therefore, suggests that there is a temperature range below the optimal operating temperature of the microbes within which higher copper extraction is achieved. Leaching below this temperature range (in this case, leaching below 17-38.6°C), results in a lower overall copper recovery. On the other hand, the leaching simulations run at temperatures higher than the optimal microbial operating temperature resulted in lower overall recovery, due to decreased microbial activity, and increased competition between the minerals.



(a) Chalcocite and covellite conversion profiles at 20°C



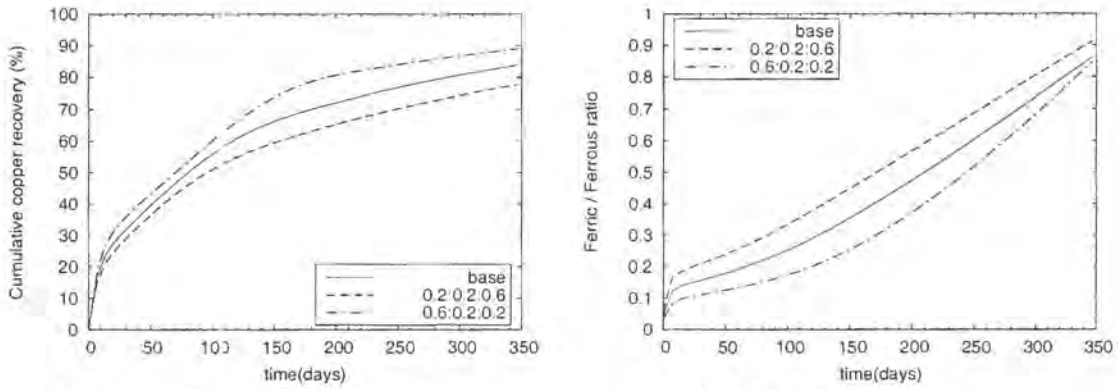
(b) Chalcocite and covellite conversion profiles at 38.6°C

Figure 4.8: Chalcocite and covellite conversion profiles in the 3mm radius size class at 20°C and 38.6°C

4.4.2 Effect of Size Class Distribution

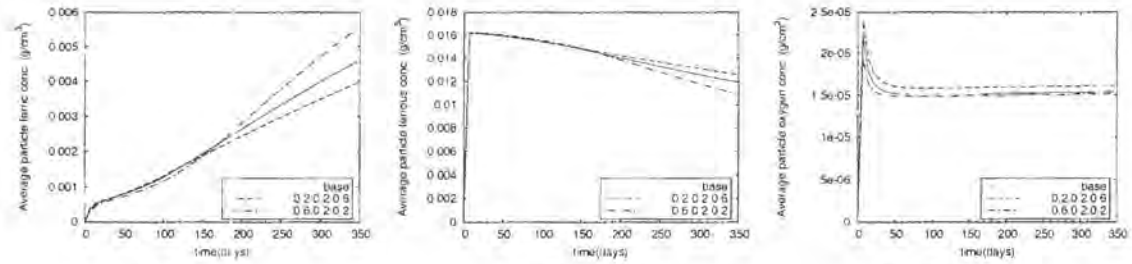
Fig.4.9 shows simulation results where the particle size classes (1mm, 3mm and 5mm radii) are equally weighted by mass (base case), and in the ratio 0.2:0.2:0.6, and 0.6:0.2:0.2, by mass.

Fig.4.9(a) shows that more copper recovery is attained when there is a higher proportion of the 1mm size class, as expected. Mineral leaching reactions in the finer particles are less limited by the diffusion of reagents to the reaction sites within the particles, compared to leaching in the coarse particles, and therefore leach to completion quicker than in the coarse particles. As a result, a higher proportion of fines leads to faster copper recovery. It is noted here that on a heap scale, the presence of very fine particles can lead to pore blocking, thereby reducing the permeability of the heap, and slowing down



(a) Copper recovery in the flowing phase (b) Bulk ferric/ferrous ratio

Figure 4.9: Effect of size class distribution on copper recovery and bulk ferric/ferrous ratio.



(a) Ferric (b) Ferrous (c) Oxygen

Figure 4.10: Average particle concentration of ferric ions, ferrous ions and dissolved oxygen for different size class distributions

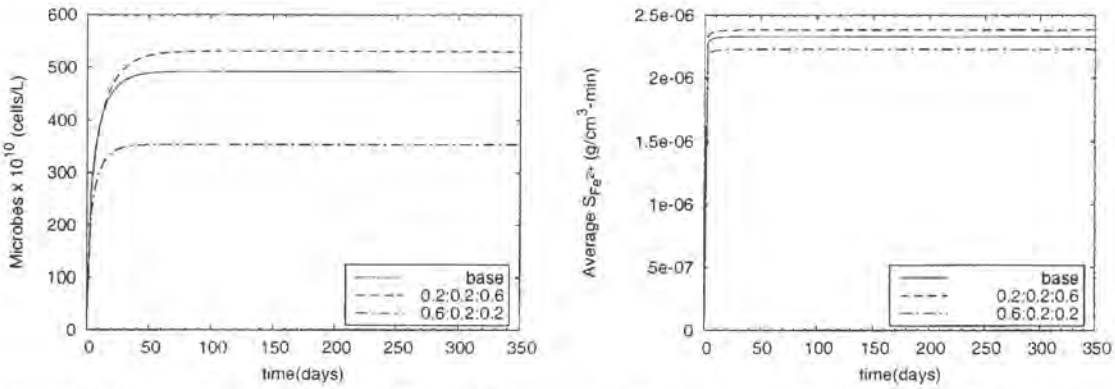
copper recovery. The ore particles are usually agglomerated to avoid very fine particles migrating in a heap.

The bulk ferric/ferrous ratio is lowest when there is a higher proportion of fines, as shown in Fig.4.9(b). The demand on ferric by the fine ore particles, and the subsequent production of ferrous, keeps the ferric/ferrous ratio in the system low for a relatively longer period of time from the start of the simulation, compared to the other size class distributions. This is clearly seen in Fig.4.10, which shows the average concentration of ferric, ferrous and oxygen in the particles, for the three size class distributions. The averages in Fig.4.10 were calculated as follows:

$$\overline{C}_i = \varepsilon_p \sum_j w_j C_{i,j} + \varepsilon_b C_{bi} \tag{4.15}$$

where i stands for the reagent, and j represents a particle size class. On completion of leaching in the finer particles, the average ferric concentration in the particles (for the 0.6:0.2:0.2 distribution) increases, as the ferric diffusing into the fine particles is not being used up in mineral leaching. On the other hand, the average ferrous concentration decreases because ferrous is no longer being produced in the fine particles. The result is that the ferric/ferrous ratio in the bulk rises more rapidly, as observed in Fig.4.9(b).

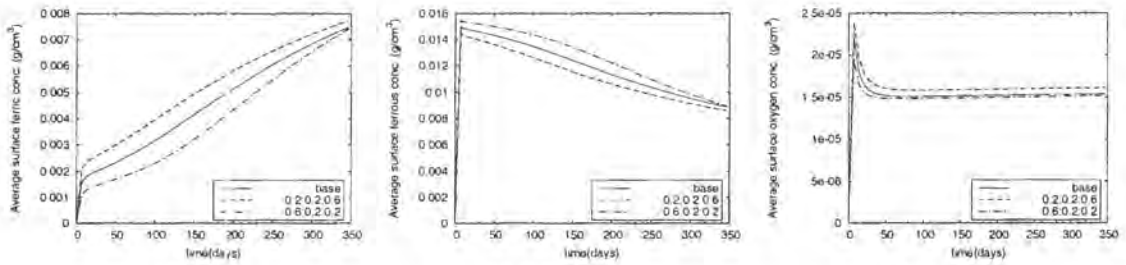
A rather puzzling result is obtained for the effect of the particle size class on microbial population. Fig.4.11 shows that a higher steady state microbial population and higher microbial ferrous oxidation rate is achieved when there is a higher proportion of coarse ore particles.



(a) Adsorbed + desorbed microbial population (b) Average microbial ferrous oxidation rate

Figure 4.11: Effect of size class distribution on microbial population and microbial ferrous oxidation rate in the agglomerate.

However, the surface and bulk concentration of oxygen is also highest when there is a higher proportion of coarse ore particles (see Fig.4.12(c) for average concentration of oxygen at the particle surface). This is counter intuitive, as one would expect a higher microbial population when there is a higher proportion of fines (due to the increased surface area for adsorption), or a lower oxygen concentration when there is a higher concentration of microbes. (As will be seen in Section 4.4.7, a lower oxygen concentration is obtained when the inlet microbial population is increased directly, keeping all other parameters constant).



(a) Ferric (b) Ferrous (c) Oxygen

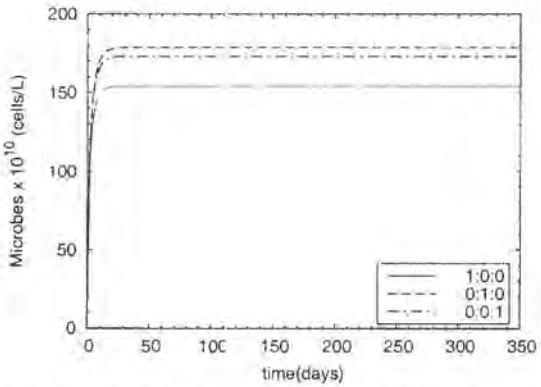
Figure 4.12: Average particle surface concentration of ferric ions, ferrous ions and dissolved oxygen for different size class distributions

In other to investigate this phenomenon, further simulations were run using the following size class distributions: 1:0:0, 0:1:0 and 0:0:1. Fig.4.13 shows the simulation results. Fig.4.13(b) shows that the surface oxygen concentration is more or less the same in these three cases. Fig.4.13(a) and Fig.4.13(c) do not seem to suggest an obvious correlation between the steady state microbial population and the bulk oxygen concentration. One must bear in mind though that most of the microbes are attached to the particle surfaces, so the surface concentration of oxygen is more significant with respect to microbial activity.

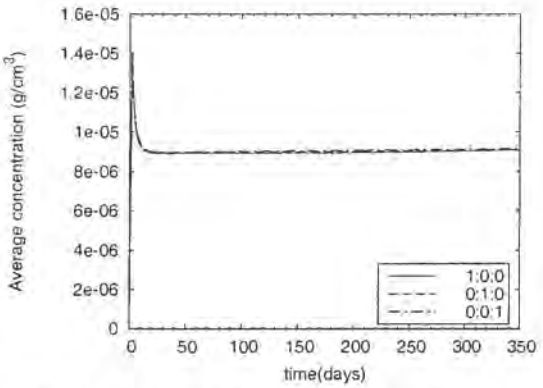
The integrity of the simulation results was queried by carrying out mass balance checks. As earlier stated, the code yields good copper mass balance results. An iron balance was also carried out. The total iron input for mineral leaching in the absence of pyrite was compared to the total iron output. (Pyrite produces extra iron, so one is able to easily match total iron input and output when there is no pyrite). This also gave good balance results, with relative error less than 0.2%. Furthermore, a careful check of the computer code did not reveal any obvious error.

The final bulk microbial population is determined by a combination of the microbial growth rate, and the Langmuir isotherm (which defines the partitioning of the microbial bulk population into adsorbed and desorbed microbes). A speculation is that in the simulations, the Langmuir isotherm somehow favors an increased microbial population for the coarser ore particles. This however does not affect simulation results for copper recovery. As previously discussed, more copper is recovered when there is a higher proportion of fines, and less when there is a higher proportion of coarse ore particles.

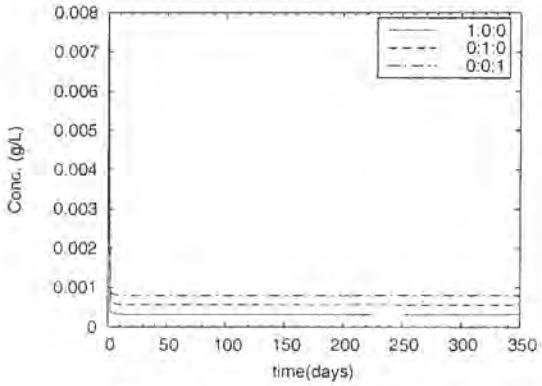
The observed anomaly in the relationship between the steady state microbial pop-



(a) Adsorbed + Desorbed microbial population



(b) Average surface oxygen concentration



(c) Bulk oxygen concentration

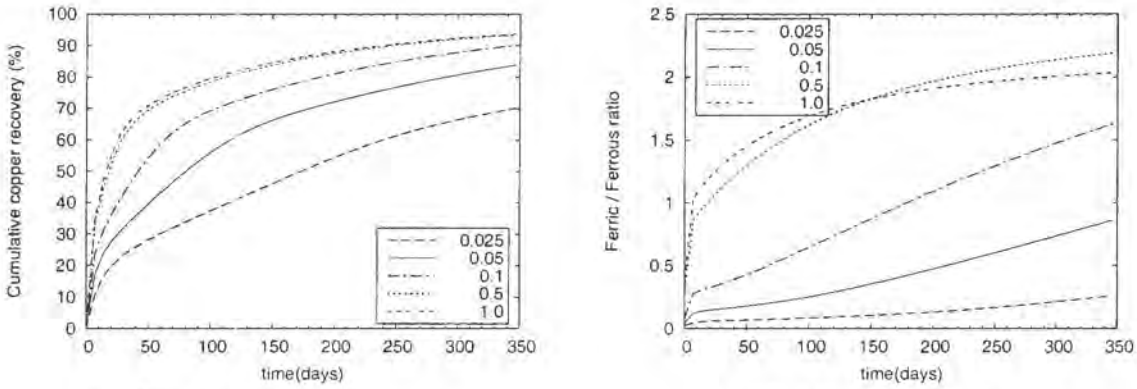
Figure 4.13: Microbial bulk population, average surface oxygen concentration, and bulk oxygen concentration, for single particle size classes.

ulation, and the particle size class distribution, is certainly an area for more thorough investigation.

4.4.3 Effect of Solution Flow Rate

The effect of the velocity of the flowing phase on copper recovery is shown in Fig.4.14(a). It can be seen from this graph that increasing the flowing phase velocity beyond a certain value does not lead to increased copper recovery.

The baseline velocity is 0.05 cm/min. Halving the flowing phase leads to a significant reduction in copper recovery, as shown in the Fig.4.14(a). This is because there is less reagent supply and less supply of microbes to the agglomerate, leading to lower rates

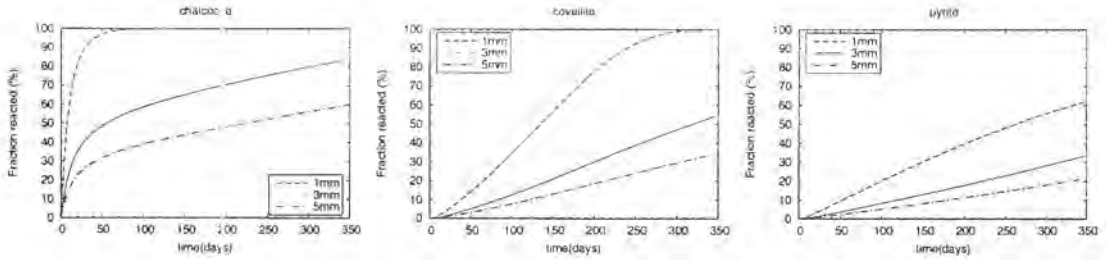


(a) Cumulative copper recovery (b) Bulk ferric/ferrous ratio.

Figure 4.14: Effect of solution flow rate on copper recovery and bulk ferric/ferrous ratio.

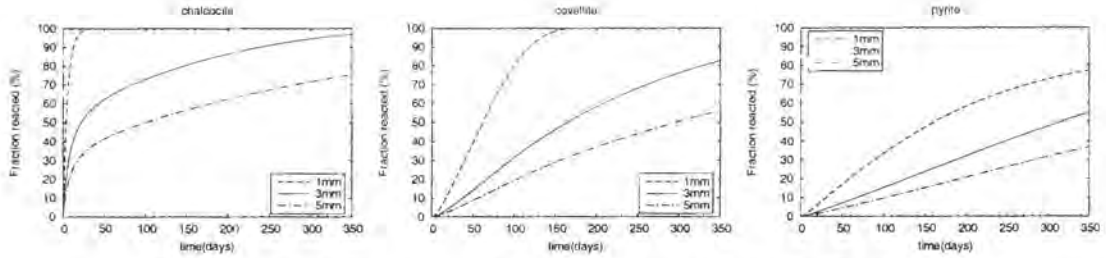
of mineral conversion (see Fig.4.15(a)). On the other hand, doubling the flowing phase velocity leads to a significant increase in copper recovery because of higher reagent supply and higher supply of microbes to the agglomerate, which results in higher mineral conversion rates (see Fig.4.15(c)).

At a velocity of 0.5 cm/min, the concentration of ferric in the system is high enough that ferric is not liniting. Hence, the rate of conversion is limited by the rate at which ferric can diffuse to, and be consumed at, the reaction sites within the particles. This is why an increase in the velocity of the flowing phase beyond this point results only in a marginal increase in copper recovery.

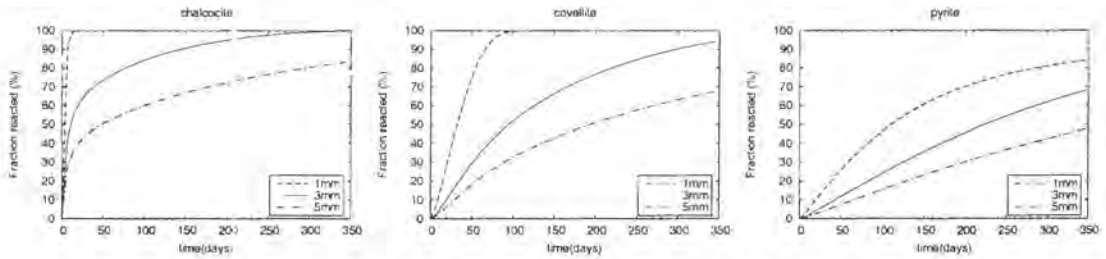


(a) Average mineral conversion profiles at u=0.025 cm/min

Figure 4.15: Average mineral conversion profiles at u=0.025 cm/min, u= 0.05 cm/min and u=0.1 cm/min



(b) Average mineral conversion profiles at u= 0.05 cm/min (base case)



(c) Average mineral conversion profiles at u=0.1 cm/min

Figure 4.15: Contd. – Average mineral conversion profiles at u=0.025 cm/min, u= 0.05 cm/min and u=0.1 cm/min.

It is interesting to note that as the flow rate is increased, the bulk ferric/ferrous ratio tends towards a lower steady state value (see Fig.4.14(b), Fig.4.16). From Fig.4.16 , it is seen that for flow rates higher than 0.5 cm/min, the final ferric/ferrous ratio decreased with increased flow rate. As highlighted in the previous paragraph, the simulation results do not show significant improvement in copper recovery beyond a flow rate of 0.5 cm/min, because the system is no longer limited by ferric concentration, but rather by the rate of diffusion and reaction of ferric in the (coarser) ore particles. It seems, therefore, that with the removal of ferric limitation in the system (due to high ferric delivery with high flow rates), the residence time of ferric and ferrous becomes, more or less, inversely proportional to the flow rate. This may be explained as follows.

The higher the flow rate, the faster complete conversion is achieved in the 1mm radius size class. Thereafter, the ferric and ferrous concentrations in this size class approach a steady state value, because these reagents are no longer active in the particles. Reagent consumption and generation is diffusion limited in the coarse ore particles. Since the diffusion rate is much slower than the flow rate, there is the tendency that, for high flow rates, most of the reagents coming into the agglomerate will be washed out. Therefore, very high flow rates lead to lower residence times of the reagents, resulting in a lower bulk

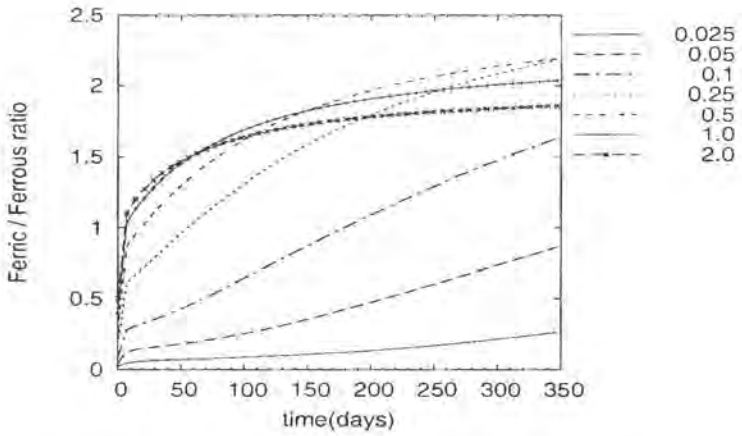


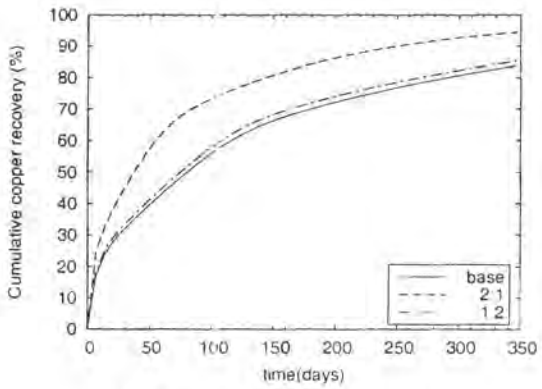
Figure 4.16: Bulk ferric/ferrous ratio for different values of the flowrate u.

ferric/ferrous ratio.

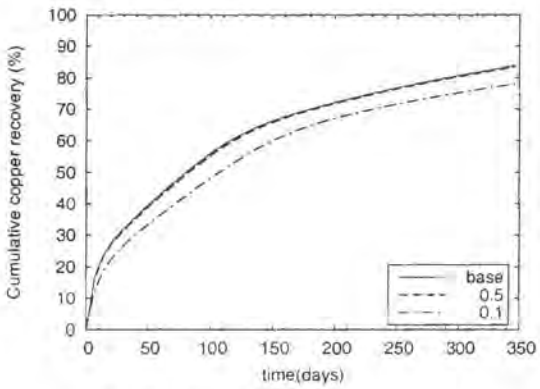
4.4.4 Effect of Inlet Substrate Concentration

The effect of ferric, ferrous, oxygen and acid concentration on copper recovery is investigated in this section. Fig.4.17 shows the effect of these substrates on copper recovery. In the simulations, the initial concentration of the substrates in the bulk were set equal to the feed substrate concentration.

As shown in Fig.4.17(a), increasing the ferric ion concentration has a more significant effect on copper recovery than increasing the ferrous ion concentration, as expected. Doubling the ferric ion concentration led to about 10% more copper recovery, while doubling the ferric ion concentration led to only about 1% more copper recovery. This is because increasing the ferrous ion concentration has an indirect effect on leaching, in that the amount of ferrous ions available for ferric regeneration by the iron oxidizing microbes is increased (unlike increasing ferric ion concentration, where the ferric is directly made available for mineral leaching). The rate of microbial ferric regeneration is influenced by other factors apart from ferrous ion concentration, such as the availability of oxygen (which is the limiting substrate in this system) and acid, and the operating temperature. Therefore, increasing the ferrous ion concentration alone does not lead to a significant increase in microbial ferric production. Although not included in the present model, ferric ion concentration also has an inhibitory effect on microbial activity (Hansford, 1997; Nemati et al., 1998; Ojumu et al., 2005; van Scherpenzeel et al., 1998).

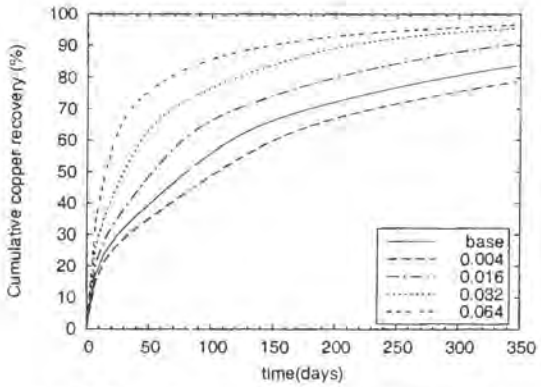


(a) Effect of ferric/ferrous ratio



(b) Effect of acid concentration (g/L)

Figure 4.17: Effect of substrate concentration on copper recovery.



(c) Effect of oxygen concentration (g/L)

Figure 4.17: Contd.

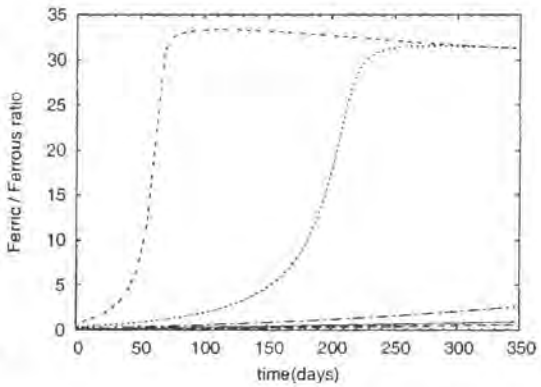


Figure 4.18: Effect of oxygen concentration on bulk ferric/ferrous ratio. (The same legend applies to Fig.4.17(c) and Fig.4.18)

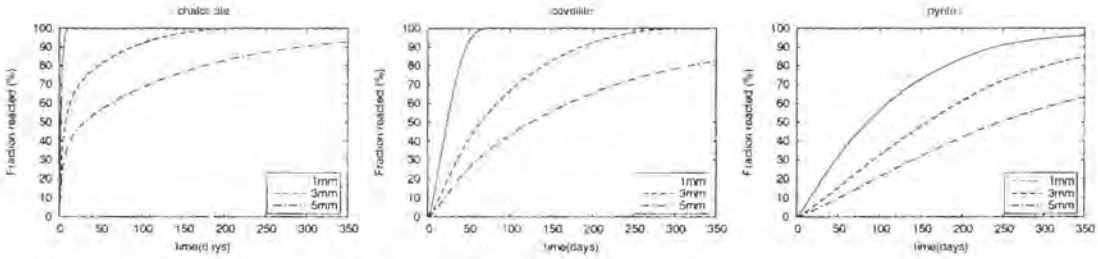
Fig.4.17(b) shows the effect of acid concentration on copper recovery. At the baseline concentration of 8 g/L, acid is in excess and so does not limit the microbial oxidation of ferric. It is seen from Fig.4.17(b) that even at acid concentration of 0.5 g/L, the effect on copper recovery is negligible. However, at a concentration of 0.1 g/L, acid becomes the limiting substrate, resulting in a reduction in copper recovery. Contrary to the low acid consumption observed from the simulation results, heaps consume large amounts of acid (Petersen, 2006). It is unclear why this is so, although a possible explanation is that the gangue material is more active than typically assumed.

Fig.4.17(c) shows the effect of different inlet oxygen concentrations on copper recovery, while Fig.4.18 shows the bulk ferric/ferrous ratios at the corresponding inlet oxygen concentrations. The baseline oxygen concentration is 0.008 g/L, which is the solubility of oxygen at ambient conditions. Fig.4.17(c) shows that operating at half the baseline oxygen concentration reduces the copper recovery considerably. Under the baseline conditions, oxygen is in limited supply compared to ferrous ions. Therefore increasing the oxygen concentration has a more significant effect on microbial ferrous oxidation rate than increasing the ferrous concentration. Fig.4.17(c) shows that increasing the inlet oxygen concentration up to 0.032 g/L significantly improves the overall copper recovery. Fig.4.18 shows that increasing the inlet oxygen concentration from 0.016 g/L to 0.032 g/L results in a much higher bulk ferric/ferrous ratio after 350 days of simulation, indicating a significant improvement in microbial activity.

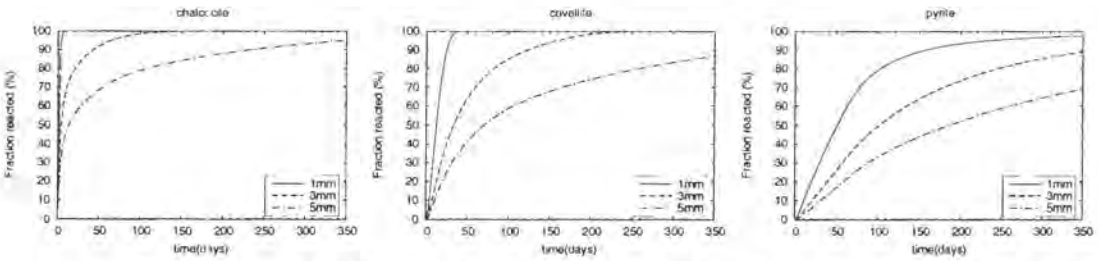
For an inlet oxygen concentration of 0.032 g/L, a sharp rise in the ferric/ferrous ratio is observed between days 150 and 250. This coincides with the completion of chalcocite leaching in the 3mm size class (see Fig.4.19(a)). The rapid chalcocite reaction in the 1mm and 3mm size classes (and also covellite conversion in the 1mm size class), consumes most of the ferric generated by the iron oxidizing microbes, and also produces ferrous ions, therefore keeping the ferric/ferrous ratio low at the initial stage of the simulation. On completion of chalcocite leaching in the 3mm size classes, covellite and pyrite leaching continues. Since these reactions are relatively slow (and mineral conversion in the 5mm size class is governed by diffusion effects), ferric consumption (and ferrous production) is now much lower, hence the ferric/ferrous ratio climbs to higher levels. At this stage, microbial oxidation is limited by low ferrous concentrations. This sort of behavior, where ferric/ferrous ratio is low at the initial stages of chalcocite leaching and rises rapidly on completion of first stage chalcocite leaching, has also been observed in column leaching experiments (Petersen and Dixon, 2003).

Fig.4.17(c) shows that increasing the inlet oxygen concentration from 0.032 g/L to 0.064 g/L leads to the recovery of a significantly higher amount of copper in a shorter period of time. A pronounced improvement in copper recovery between days 0–150 is observed, mostly from chalcocite and covellite leaching in the 1mm and 3mm size classes (see Fig.4.19(b)). Subsequent copper recovery is governed by diffusion distance in the coarse ore particles rather than reagent concentration, which is why the overall copper recovery after 350 days is only about 1% higher.

It is noted that dissolved oxygen concentrations as high as 0.016 g/L may not be



(a) Inlet oxygen concentration = 0.032 g/L



(b) Inlet oxygen concentration = 0.064 g/L

Figure 4.19: Mineral conversion profiles for inlet oxygen concentration of 0.032 g/L and 0.064 g/L.

attainable under heap bioleaching conditions, owing to the fact that the heap usually reaches high temperatures (50–60°C for pyrite containing heaps), and oxygen solubility decreases with increasing temperature. Also, in a heap, oxygen must undergo a mass transfer step from the gaseous phase to the liquid phase, and this step can be rate-limiting (Bouffard and Dixon, 2003). In this model, a fixed concentration of dissolved oxygen in the incoming liquid phase was assumed. The inclusion of an oxygen gas-liquid mass transfer step should be considered in further modelling work.

With respect to the baseline conditions, the substrates that have the most effect on improving copper recovery are, (in decreasing order of effect), ferric ions, dissolved oxygen, ferrous ions and acid.

4.4.5 Effect of Mass Transfer Parameters

The mass transfer parameters in this model are k_{sb}, the mass transfer coefficient at the particle surface/bulk interface (Eq.(3.3)), and k^{*}_{bf}, the mass transfer rate at the bulk/flowing phase interface (Eq.(3.5)). The mass transfer coefficient is a function of bulk flow conditions and the local geometry, and is usually a difficult parameter to determine

accurately (Petersen, 1998). It appears in correlations for dimensionless numbers in heat transfer theory, and due to the analogy between heat and mass transfer, correlations established for heat transfer can also be used for mass transfer problems. Commonly, the mass transfer parameter appears in the Sherwood number Sh:

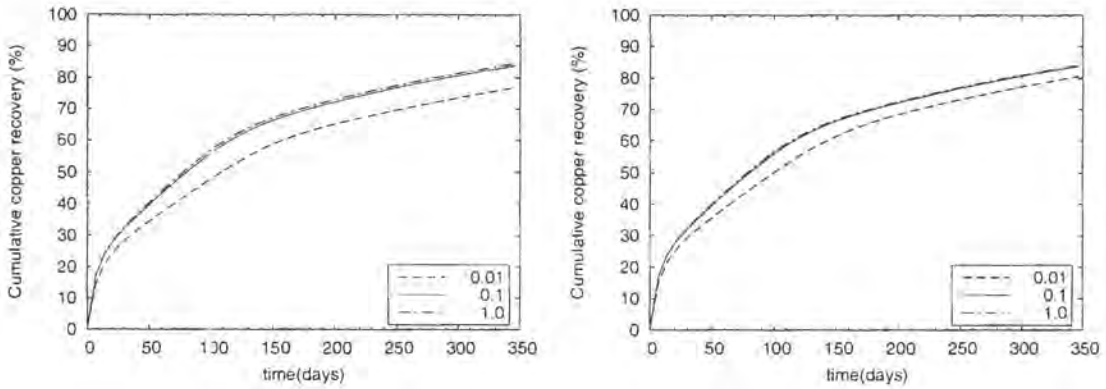
$$Sh = \frac{k_c d}{D^*} \tag{4.16}$$

or the in terms of the Chilton-Colburn factor j_D:

$$j_D = \frac{k_c}{v}\left(\frac{\mu}{\rho D^*}\right)^{\frac{2}{3}} \tag{4.17}$$

(Petersen, 1998). In Eq.(4.16) and Eq.(4.17). k_c is the mass transfer coefficient, d is a characteristic length, D^* is the molecular diffusivity, μ is the fluid viscosity, ρ is the fluid mass density, and v is the seepage (linear pore) velocity.

In this work, constant values were assumed for the two mass transfer parameters, $k_{\rm sb}$ and $k^*_{\rm bf}$. The sensitivity of simulation results (in terms of copper recovery in the flowing phase), to changes in the value of $k_{\rm sb}$ and $k^*_{\rm bf}$, is illustrated in Fig.4.20. The simulations were run using the baseline parameters, and changing only the respective mass transfer parameters. The baseline value of $k_{\rm sb}$=0.1 cm(min^{-1}), and $k^*_{\rm bf}$=0.1 min^{-1}.



(a) Effect of $k_{\rm sb}$ on copper recovery

(b) Effect of $k^*_{\rm bf}$ on copper recovery

Figure 4.20: Effect of $k_{\rm sb}$ and $k^*_{\rm bf}$ on copper recovery.

Fig.4.20 shows that a tenfold increase in $k_{\rm sb}$ and $k^*_{\rm bf}$ above the baseline values has a negligible effect on the calculated copper recovery, whereas the converse is the case for a tenfold decrease in these parameters. In other words, the baseline parameters used in

the simulation represent a situation where there is little mass transfer limitation at the particle surface/bulk interface, and at the bulk/flowing phase interface.

Decreasing the mass transfer parameters leads to lower copper recovery because mass transfer limitation has been introduced at the respective interfaces. From Fig.4.20, a tenfold decrease in k_{sb} leads to less copper recovery compared with a tenfold decrease in k^*_{bf}.

Comparing Fig.4.20(a) and Fig.4.20(b), it is seen that copper recovery is more sensitive to the k_{sb} parameter than the k^*_{bf} parameter.

4.4.6 Effect of Arrhenius Rate Constant

The intrinsic rate of chalcocite reaction is high compared to that of covellite and pyrite, hence the effect of its Arrhenius rate constant is not included in this investigation.

Increasing the rate constant of covellite conversion by a factor of 2 increases copper recovery, as shown in Fig.4.21. From Fig.4.21, it is seen that copper recovery is considerably higher between days 20 and 130, for a higher covellite rate constant, compared to the base case. This time interval is dominated by covellite leaching in the 1mm size class (see Fig.4.22(b)). On completion of chalcocite and covellite leaching in the 1mm size class (around day 130), copper recovery slows down, and the overall additional copper recovery after 350 days of leaching is about 1%. This is because subsequent leaching in 3mm and 5mm size classes is dominated by the effect diffusion distance, as discussed in Section 4.2.

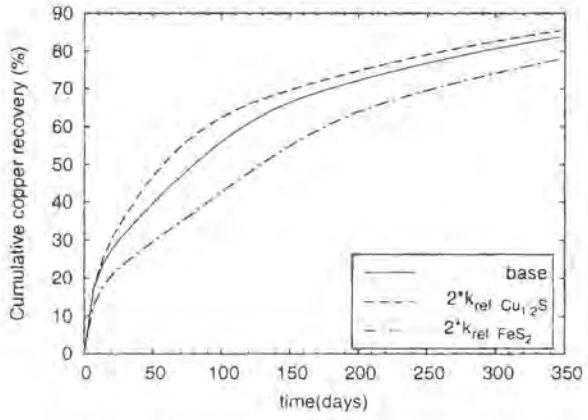
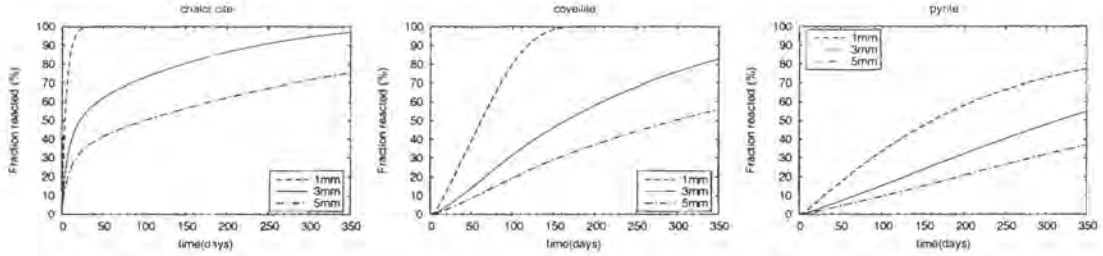


Figure 4.21: Effect of covellite and pyrite rate constant on copper recovery.

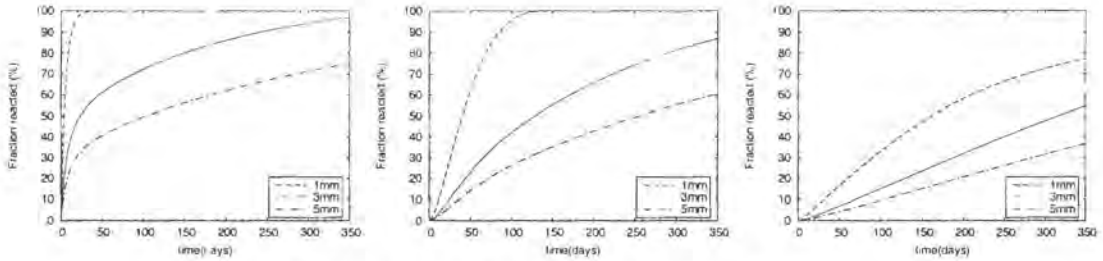
Increasing the pyrite rate constant by a factor of 2 results in a marked decrease in

copper recovery, as shown in Fig.4.21. 1 mol of pyrite reacts with 14 moles of ferric ions (see Eq.(4.3), where β=0). Therefore increasing the rate of pyrite reaction increases the rate of ferric consumption by pyrite, limiting the amount of ferric available for chalcocite and covellite leaching.

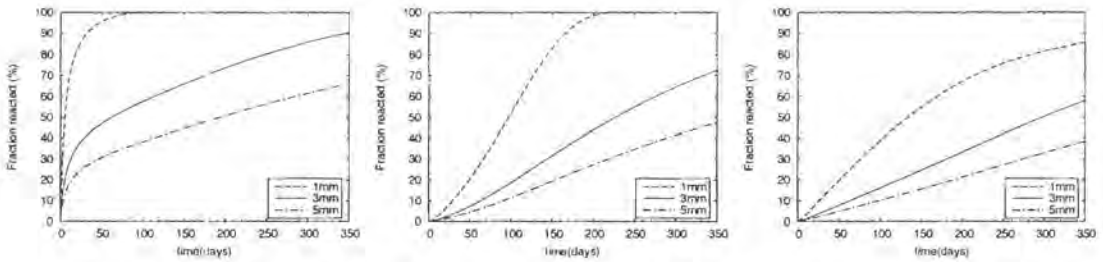
Fig.4.22 shows the average mineral conversion profiles for the cases discussed in this section. From these graphs, it is seen that doubling the covellite rate constant increases covellite conversion, but has a small effect on chalcocite and pyrite conversion. On the other hand, doubling the pyrite rate constant has the pronounced effect of slowing down chalcocite and covellite conversion.



(a) Average mineral conversion for base case.



(b) Average mineral conversion for $2^{*}k_{\mathrm{ref,Cu_{1.2}S}}$.

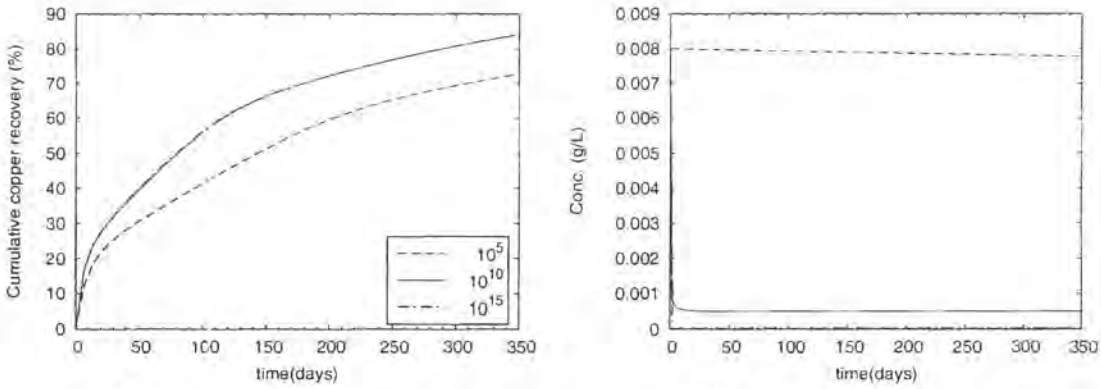


(c) Average mineral conversion for $2^{*}k_{\mathrm{ref,FeS_2}}$

Figure 4.22: Average mineral conversion profiles for baseline, $2^{*}k_{\mathrm{ref,Cu_{1.2}S}}$ and $2^{*}k_{\mathrm{ref,FeS_2}}$.

4.4.7 Effect of Inlet Microbial Population

The effect of the inlet microbial population of iron oxidizing microbes on copper recovery is shown in Fig.4.23(a). The baseline microbial population is 10^{10} cells/L. In the simulations, the initial desorbed microbial concentration was set equal to the feed microbial concentration. The legend for Fig.4.23(a) also applies to Fig.4.23(b).



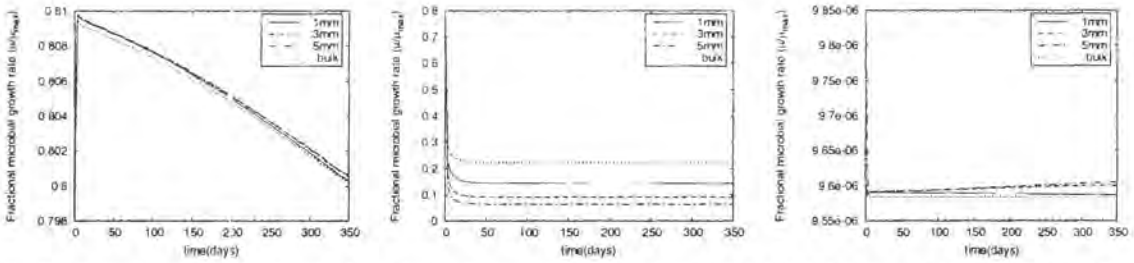
(a) Cumulative copper recovery (b) Bulk oxygen concentration

Figure 4.23: Effect of inlet microbial population on copper recovery and bulk oxygen concentration.

As shown in Fig.4.23(a), increasing the inlet microbial population to 10^{15} cells/L (while maintaining all other baseline parameters), does not lead to a noticeable increase in copper recovery. The increased microbial population leads to increased demand for ferrous, dissolved oxygen and acid. Oxygen is in limited supply with respect to ferrous and acid (see Fig.4.23(b)), and the microbes can oxidize ferrous only to the extent that dissolved oxygen is available. Therefore increasing the microbial population without a corresponding increase in dissolved oxygen concentration does not lead to an appreciable increase in microbial ferric ion production. Consequently, leaching is not enhanced.

Decreasing the inlet microbial population leads to a reduction in copper recovery. At an inlet microbial concentration of 10^5 cells/L, the copper recovery is comparable to the recovery in an abiotic system (compare Fig.4.23(a) and Fig.4.5(a)), implying that copper recovery is not being enhanced by the presence of the microbes. The microbial population is too low to have a significant effect on increasing the ferric concentration.

Fig.4.24 shows the fractional microbial growth rates, $\mu/\mu_{\max}$, for the three microbial populations investigated. For the low microbial population (10^5 cells/L), the fractional

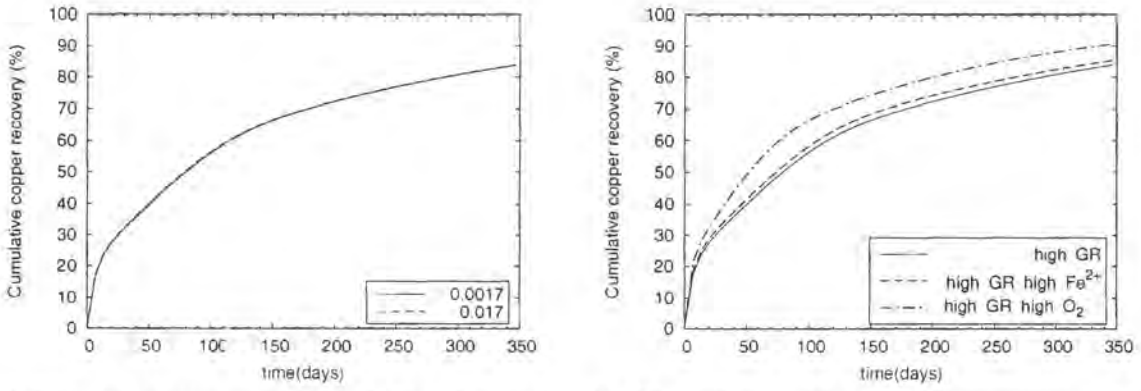


(a) Inlet population=10^5 cells/L (b) Inlet population=10^{10} cells/L (c) Inlet population=10^{15} cells/L

Figure 4.24: Fractional microbial growth rates, $\mu/\mu_{\max}$, for different inlet microbial populations.

growth rate is high. However, the microbial population is too low for appreciable ferric production, and the maximum growth rate, $\mu_{\max}$, is such that the microbial population does not increase significantly within the leaching period simulated. On the other hand, for the high microbial population (10^{15} cells/L), the fractional microbial growth rate is very low, so that the overall microbial ferric oxidation is comparable with the oxidation by the baseline population (10^{10} cells/L).

4.4.8 Effect of Maximum Microbial Growth Rate



(a) Cumulative copper recovery for baseline and high $\mu_{\max}$ (b) Effect of $\mu_{\max}$ for high ferrous concentration and high oxygen concentration

Figure 4.25: Effect of $\mu_{\max}$ on copper recovery.

Fig.4.25(a) shows that increasing the maximum microbial growth rate $\mu_{\max}$, while maintaining all other baseline parameters, has negligible effect on the overall copper re-

covery. On the other hand, operating at a high growth rate and increasing the inlet concentration of ferrous or oxygen (in this case by a factor of 2) increases copper recovery, as shown in Fig.4.25(b). This suggests that using faster growing microbes will not necessarily improve initial copper recovery under substrate limiting conditions, and therefore the emphasis should rather lie on improving the substrate levels. However, if the initial microbial population is low, and there is an adequate amount of substrate (for example, the situation for the 10^5 cells/L inlet population discussed in Section 4.4.7), then a high growth rate is beneficial, as the microbes are able to attain a high population in a shorter period of time.

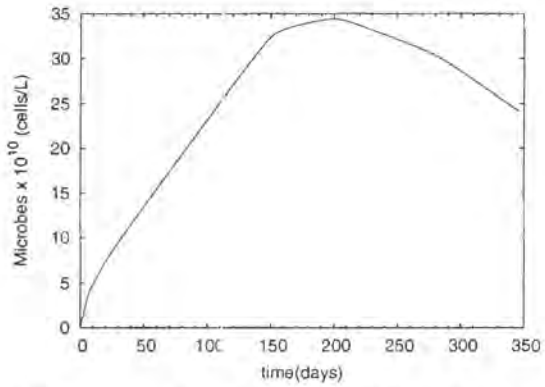
4.4.9 Presence of Sulfur Oxidizers

Sulfur oxidizers react with elemental sulfur according to Eq.(4.6) to produce sulfuric acid. Elemental sulfur is a product in reactions Eq.(4.2) and Eq.(4.3) (when $\beta > 0$). Fowler and Crundwell (1999) report that in the absence of bacteria, a product layer of sulfur is formed in the ferric leaching of zinc sulfide, which acts as a diffusion barrier for ferric ions. In the presence of bacteria, sulfur is oxidized, no product layer is observed, and reaction with ferric ions becomes the rate-limiting step.

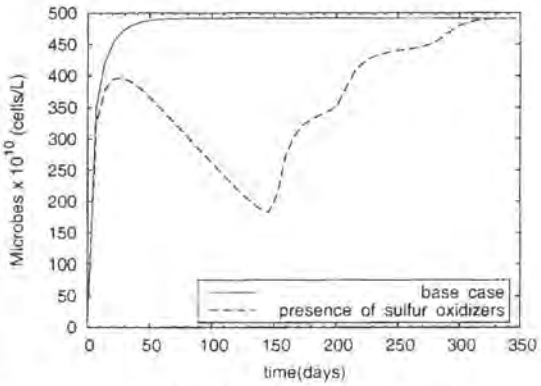
In this section, sulfur formation was not modelled as a diffusion barrier to ferric ions. The action of sulfur oxidizing bacteria is then primarily the production of acid. The assumption made in the model is that the desorbed sulfur oxidizers do not grow, because there is no elemental sulfur in the stagnant liquid. On the other hand, the growth rate of the attached sulfur oxidizers is limited by the rate of production of elemental sulfur from covellite leaching.

The effect of a comparable population of iron and sulfur oxidizers on the leaching process was investigated by introducing additional elemental sulfur into the system. An initial concentration of 0.1 g/g_{ore} of elemental sulfur was specified at the surface of the ore particles, and inlet populations of 10^9 cells/L and 10^{10} cells/L for sulfur and iron oxidizers respectively were used. Simulation results are shown in Fig.4.26.

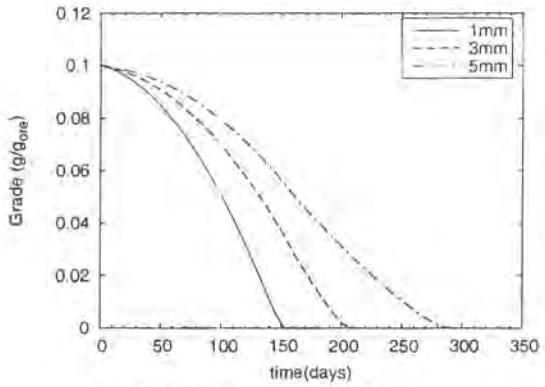
For the first ten days of the simulation, there is an increase in the population of both iron and sulfur oxidizers. The iron oxidizers rise to a higher initial population due to a higher growth rate, and the availability of ferrous ions. Subsequently, the population of iron oxidizers falls as competition for oxygen between the two microbial species increases. On the other hand, the competition for oxygen leads to a reduced rate of increase of the



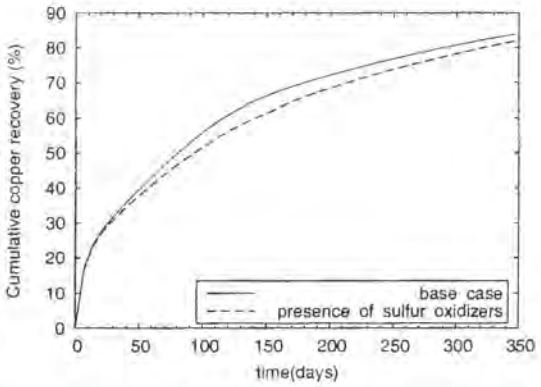
(a) Population of (adsorbed + desorbed) sulfur oxidizers



(b) Population of (adsorbed + desorbed) iron oxidizers



(c) Surface sulfur concentration



(d) Copper recovery

Figure 4.26: Microbial population, surface sulfur concentration, and copper recovery in the presence of sulfur oxidizers.

sulfur oxidizing population.

With the exhaustion of the elemental sulfur available at the surface of a particle size class, the rate of growth of sulfur oxidizers reduces significantly, accompanied by an increase in the population of iron oxidizers (compare Fig.4.26(a), (b) and (c)). Eventually, all the elemental sulfur on the particle surfaces is consumed after about 290 days, and the attached sulfur oxidizers can no longer grow. On the other hand, the population of iron oxidizers is able to reach a steady state (the same steady state achieved in the absence of sulfur oxidizers). On the whole, copper recovery is less in the presence of sulfur oxidizers compared to the base case (see Fig.4.26(d)), due to the lower population of the iron oxidizers during the simulation.

It is noted that 0.1 g/g_{ore} is a fairly high concentration of elemental sulfur for a heap bioleaching system. This concentration was chosen in order to investigate the effects of a comparable population of iron and sulfur oxidizers on the leaching process. Initial simulations run using baseline parameters, and both sulfur and iron oxidizers at inlet populations of 10^9 cells/L and 10^{10} cells/L respectively encountered numerical errors, with the population of the sulfur oxidizers going into negative values. This is due to the rapid depletion of sulfur oxidizers in the early stages of the simulation, owing to an insufficient amount of elemental sulfur at the particle surfaces to support the growth of the attached microbes. The initial shortage of elemental sulfur at the particle surfaces is because covellite conversion in that region is delayed in the early stages of the leaching simulation by the conversion of the more reactive chalcocite.

In reality, it could be that the sulfur oxidizers do not actually attach to the ore particles in the absence of elemental sulfur. However, the model simulates this by first calculating the expected population of attached sulfur oxidizers from the desorbed population using the Langmuir isotherm, and then calculating the change in this adsorbed population from Eq.(3.8) (which is in terms of the microbial growth and death rates). The expected numerical result is that the population of sulfur oxidizers will decrease to zero in the absence of elemental sulfur to sustain the attached population. This insufficient elemental sulfur at the initial stages of the simulation means that the microbial growth rate is close to zero, and this, coupled with a large inlet population of sulfur oxidizers and a negative death rate, leads to a rapid decrease in the population of sulfur oxidizers initially, which the numerical method did not accurately capture. The initial rapid phase will require very small timesteps for the numerical method to accurately represent the change in the population of adsorbed sulfur oxidizers in this phase. Further simulations run with smaller inlet populations of sulfur oxidizers gave results where the population of sulfur oxidizers decreased to negative values in the first few days of the simulation, and then rose to positive values as the simulation progressed. The initial dip of the sulfur oxidizing population was less negative as smaller inlet populations of sulfur oxidizers were used.

The simulation results presented in Fig.4.26 give an indication of the interaction between iron and sulfur oxidizers in heap bioleaching, and the influence this as on copper recovery. However, the microbial phenomena are much more complex. For instance, *At. ferrooxidans*, an important mesophile in heap bioleaching, can grow on both ferrous ions and elemental sulfur (Rawlings, 2002). The microbial model implemented here is one of several proposed for microbial activity in heap bioleaching. As highlighted in the liter-

ature review, mathematical modelling of microbial processes in heap bioleaching is an active area of study.

4.5 Conclusion

In this chapter, numerical simulations were carried out to investigate the effect of selected model parameters on copper recovery for a theoretical case study ore containing chalcocite and pyrite, in the presence of iron oxidizing microbes. The ore particles were grouped into three size classes of radii 1mm, 3mm and 5mm.

With respect to copper recovery, the sensitivities of the following parameters were investigated: temperature, particle size distribution, solution flow rate, inlet substrate concentration, mass transfer parameters, Arrhenius rate constant, inlet microbial population, maximum microbial growth rate, and presence of sulfur oxidizers. The base case was operating under oxygen-limiting conditions.

A significant finding from the simulation results is the possibility of higher copper recovery by operating at a temperature range below the optimum temperature of the iron oxidizing microbes due to the competitive leaching of minerals and lower microbial ferric generation at higher temperatures. This finding should be verified experimentally.

The parameters that had most effect on increasing copper recovery above the base case value were the inlet ferric concentration, inlet dissolved oxygen concentration, flow rate, and increased proportion of fine ore particles. Of these parameters, the first three deal with the delivery of reagents to the agglomerate, suggesting this as the key factor to improving copper recovery at the agglomerate level.

An unexpected simulation result was observed in the effect of particle size distribution on the steady state microbial population. The cause of this could not be ascertained, and the speculation is that it is due to the implementation of the Langmuir isotherm. This is certainly an area for more thorough investigation. It is noted that despite this unexplained result, the simulation results showed the correct trend with respect to the effect of particle size distribution on copper recovery.

CHAPTER 5

Closure

5.1 Concluding Discussion

This thesis has been concerned with the development of a mathematical model for heap bioleaching at the agglomerate scale, an intermediate scale between the particle scale and the bulk scale. The agglomerate was defined as a unit volume of a heap that comprises a solid phase (a size distribution of ore particles), a liquid phase (stagnant and flowing leaching solution, which contains dissolved solutes, attached and planktonic microbes) and a gas phase (flowing air and air pockets). The proposed agglomerate model provides a framework for the systematic integration of particle scale processes into bulk scale models. It can also be queried to gain insight into heap bioleaching at the agglomerate scale, or even applied to study bioleaching in short laboratory columns.

The agglomerate model was developed in terms of a single unit volume of the heap. Simplifying assumptions were made in the model development. Notable among them are isothermal agglomerate conditions, the absence of a gas phase (but a fixed concentration of dissolved oxygen in the incoming flowing liquid phase), and well mixed conditions in the stagnant liquid phase.

With the help of this model, the agglomerate scale bioleaching of a theoretical case study ore containing chalcocite and pyrite, in the presence of iron oxidizing microbes, was investigated. At the particle scale, the successive nature of the first and second stage chalcocite leaching was observed, with covellite conversion only in regions where chalcocite had been depleted (see Fig.4.2). Base case simulations showed improved copper recovery in the presence of iron oxidizing microbes (see Fig.4.5). It also showed an initial rapid copper extraction rate which gradually declined as diffusion limitation effects set in (see Fig.4.1(a)). At suitably high inlet oxygen concentrations, a rapid rise in ferric/ ferrous ratio was observed on completion of first stage chalcocite leaching in the finer ore particles (see Fig.4.18). The observations stated above have also been reported in an experimental study of the column leaching of an ore containing chalcocite and pyrite in the presence

of iron oxidizing microbes (Petersen and Dixon, 2003). Hence, the base case simulation results show that the agglomerate model is able to predict expected qualitative behavior.

Model sensitivity studies were carried out to test the sensitivity of the cumulative copper recovery to selected parameters. The parameters investigated were temperature, particle size distribution, solution flow rate, inlet substrate concentration, mass transfer parameters, Arrhenius rate constant, inlet microbial population, maximum microbial growth rate, and presence of sulfur oxidizers. The sensitivity studies showed that the following parameters had the largest effect on increasing copper recovery above the base case value: flow rate, inlet ferric concentration, inlet oxygen concentration, and the proportion of fine ore particles.

The study of model sensitivity to temperature indicated the possibility of higher copper recovery by operating in a certain temperature range below the optimum temperature of the iron oxidizing microbes, due to competitive leaching of minerals and lower microbial ferric generation at higher temperatures.

The interaction between iron and sulfur oxidizers was studied by specifying an initial surface concentration of elemental sulfur for the ore particles. The simulation results showed that the introduction of sulfur oxidizers led to competition for oxygen between the sulfur and iron oxidizers, leading to a lower population of iron oxidizers compared to the base case simulation, and hence lower copper recovery. Initial simulations using iron and sulfur oxidizers without the introduction of additional elemental sulfur encountered numerical inaccuracies, due to severe elemental sulfur limitations.

5.2 Recommendations for Further Work

On the premise of the work done so far on the agglomerate model, a logical step will be the application of this model to a physical case. The aforementioned model prediction of the response of copper recovery to temperature should be tested by performing experiments.

The relationship between the particle size distribution and the steady state microbial population requires further investigation. The simulation results showed that an increased proportion of coarse ore particles led to an increased steady state bulk population of microbes, which is clearly counter-intuitive. A conclusive explanation for this behavior could not be given at this stage, although a speculation is that it is related to the current implementation of the Langmuir isotherm.

As has been previously highlighted, two significant processes not included in the current model are oxygen gas-liquid transfer, and temperature variation. The proposed agglomerate model can be made more realistic by including these two processes. The inclusion of the oxygen gas-liquid transfer will require the explicit addition of a gas phase. This can be achieved in the current framework by introducing a flowing gas phase of volume fraction ε_g (such that $\varepsilon_p + \varepsilon_b + \varepsilon_f + \varepsilon_g = 1$), and writing mass balance equations for oxygen similar to Eq.(3.4) and Eq.(3.6), where the rate of oxygen exchange between the flowing gas phase and the stagnant liquid phase is a function of the gas-liquid mass transfer coefficient, the oxygen concentration in the flowing gas and the stagnant liquid phases, and Henry's coefficient. This is typically written as:

$$k_L a(C_{b,O_2} - H_e C_{g,O_2}) \tag{5.1}$$

where C_{b,O_2} and C_{g,O_2} are the oxygen concentration in the liquid (bulk) and gas phases respectively, $k_L a$ is the mass transfer coefficient, and H_e is Henry's constant (which is dependent on temperature). The incorporation of temperature variation into the current model will require the addition of a heat balance equation.

An important step from the present agglomerate model is its integration into a bulk scale model. A suggestion is to assume that the unit volumes are stacked to form a unit heap column, and incorporating the interaction between neighboring unit volumes in the unit heap column, and bulk flow phenomena. Further development from this point will require a description of the interaction between adjacent heap columns.

References

R.W. Bartlett. Simulation of ore heap leaching using deterministic models. *Hydrometallurgy*, 29:231–260, 1992a.

R.W. Bartlett. *Solution Mining-Leaching and Fluid Recovery of Materials*. Gordon and Breach Science Publishers, 1992b.

C.R. Bennet, M. Cross, T.N. Croft, J.L. Uhrie, C.R. Green, and J.E. Gebhardt. A comprehensive stockpile leach model: background and model formulation. In C.A. Young, A.M. Alfantazi, C.G. Anderson, D.B. Dreisinger, B. Harris, and A. James, editors, *Hydrometallurgy 2003*, volume 1, pages 315–328. TMS, 2003.

S.C. Bouffard and D.G. Dixon. Mathematical modelling of pyritic refractory gold ore heap biooxidation: model development and isothermal column simulations. In C.A. Young, A.M. Alfantazi, C.G. Anderson, D.B. Dreisinger, B. Harris, and A. James, editors, *Hydrometallurgy 2003*, volume 1, pages 275–288. TMS, 2003.

R.L. Braun, A.E. Lewis, and M.E. Wadsworth. In-place leaching of primary sulphide ores: Laboratory leaching data and kinetics model. *Solution Mining Symposium, AIME*, pages 295–323, 1974.

J.A. Brierley. Heap leaching of gold-bearing deposits: theory and operational description. In Rawlings (1997a), chapter 5, pages 103–115.

J.A. Brierley and C.L. Brierley. Present and future commercial applications of biohydrometallurgy. *Hydrometallurgy*, 59:233–239, 2001.

E. Cariaga, F. Concha, and M. Sepúlveda. Flow through porous media with applications to heap leaching of copper ores. *Chemical Engineering Journal*, 111:151–165, 2005.

J.M. Casas, J. Martinez, L. Moreno, and T. Vargas. Bioleaching model of a copper-sulphide ore bed in heap and dump configurations. *Metallurgical and Materials Transactions B*, 29:899–909, 1998.

F.K. Crundwell. Physical chemistry of bacterial leaching. In Rawlings (1997a), chapter 9, pages 177–200.

G.B. Davis, G. Doherty, and A.I.M. Ritchie. A model of oxidation in pyritic mine Wastes: part 2 – Comparison of numerical and approximate solutions. *Applied Mathematical Modelling*, 10:323–329, 1986.

G.B. Davis and A.I.M. Ritchie. A model of oxidation in pyritic mine wastes: part 1 – Equations and approximate solution. *Applied Mathematical Modelling*, 10:314–322, 1986.

G.B. Davis and A I.M. Ritchie. A model of oxidation in pyritic mine wastes: part 3 – Import of particle size distribution. *Applied Mathematical Modelling*, 11:417–422, 1987.

D.G. Dixon. Analysis of heat conservation during copper sulphide heap leaching. *Hydrometallurgy*, 58:27–41, 2000.

D.G. Dixon. Heap leach modelling – the current state of the art. In C.A. Young, A.M. Alfantazi, C.G. Anderson, D.B. Dreisinger, B. Harris, and A. James, editors, *Hydrometallurgy 2003*, volume 1, pages 289–314. TMS, 2003.

D.G. Dixon and J.L. Hendrix. Theoretical basis for variable order assumption in the kinetics of leaching of discrete grains. *AICHE J*, 39:904–907, 1993.

D.G. Dixon and J. Petersen. Comprehensive modelling study of chalcocite column and heap bioleaching. In P.A. Riveros, D. Dixon, D. Dreisinger, and J. Menacho, editors, *Copper 2003-Hydrometallurgy of Copper (Book 2)*, volume 6, pages 493–516. CIM, 2003.

T.A Fowler and F.K. Crundwell. Leaching of zinc sulfide by *Thiobacillus ferrooxidans*: bacterial oxidation of the sulfir product layer increaseeeees the rate of zinc sulfide dissolution at high concentrations of ferrous ions. *Applied Environmental Microbiology*, 65:5285–5292, 1999.

P.D. Franzmann, C.M. Haddad, R.B. Hawkes, W.J. Robertson, and J.J. Plumb. Effects of temperature on the rates of iron and sulfur oxidation by selected bioleaching *Bacteria* and *Archaea*: Application of the ratkowsky equation. *Minerals Engineering*, 18:1304–1314, 2005.

D.G. Fredlund and H. Rahardjo. *Soil Mechanics for unsaturated soils*. Wiley, 1993.

G.S. Hansford. Recent developments in modelling the kinetics of bioleaching. In Rawlings (1997a), chapter 8, pages 153–175.

D.G. Karamanev. Model of the biofilm structure of *Thiobacillus ferrooxidans*. *Journal of Biotechnology*, 20:51–64, 1991.

P. Kinnunen. *High-rate ferric sulfate generation and chalcopyrite concentrate leaching by Acidophilic microorganisms*. PhD thesis, Tampere University of Technology, Finland, 2004.

M.J. Leahy, M.P. Schwarz, and M.R. Davidson. An air sparging cfd model for heap bioleaching of copper sulphide. In *Third International Conference on CFD in the Minerals and Process Industries*, pages 581–586, Australia, 2003. CSIRO.

M.J. Leahy, M.R. Davidson, and M.P. Schwarz. A model for heap bioleaching of chalcocite with heat balance: Bacterial temperature dependence. *Minerals Engineering*, 18:1239–1252, 2005a.

M.J. Leahy, M.R. Davidson, and M.P. Schwarz. A two-dimensional cfd model for heap bioleaching of chalcocite. *ANZIAM J*, 46(E):C439–C457, 2005b.

O. Levenspiel. *Chemical Reaction Engineering*. John Wiley & Sons, Inc., second edition, 1972.

L. Moreno, J. Martinez, and J. Casas. Modelling of bioleaching copper sulphide in heaps or dumps. In R. Amils and A. Ballester, editors, *Biohydrometallurgy Symposium IBS 99*, pages 443–451, 1999.

M. Nemati, S.T.L. Harrison, G.S. Hansford, and C. Webb. Biological oxidation of ferrous sulphate by *Thiobacillus ferrooxidans*: a review on the kinetic aspects. *Biochemical Engineering Journal*, 1:171–190, 1998.

H.J. Neuburg, J.A. Castillo, M.N. Herrera, J.V. Wiertz, T. Vargas, and R. Badilla-Ohlbaum. A model for the bacterial leaching of copper sulphide ores in pilot-scale columns. *Internationall Journal of Mineral Processing*, 31:247–264, 1991.

M.J. Nicol and C.R.S. Needes. Electrochemical model for the leaching of uranium dioxide: 1. acidic media. In A.R. Burkin, editor, *Leaching and reduction in Hydrometallurgy*, pages 1–9. IMM, Camelot press, 1975.

T.V. Ojumu, J. Petersen, G.E. Searby, and G.S. Hansford. A review of rate equations proposed for microbial ferrous-iron oxidation with a view to application to heap bioleaching.

In S.T.L. Harrison, D.E. Rawlings, and J. Petersen, editors, *16th International Biohydrometallurgy Symposium, IBS2005*, pages 85–93, Cape Town, South Africa, 2005.

G.J. Olson, J.A. Brierley, and C.L. Brierley. Bioleaching review part B: Progress in bioleaching: applications of microbial processes by the minerals industries. *Applied Microbiology Biotechnology*, 63:249–257, 2003.

S. Orr. Enhanced heap leaching – part 1: Insights. *Mining Engineering*, 54, part 9:49–56, September 2002.

G. Pantelis and A.I.M Ritchie. Macroscopic transport mechanisms as a rate-limiting factor in dump leaching of pyritic ores. *Applied Mathematical Modelling*, 15:136–143, 1991.

G. Pantelis and A.I.M Ritchie. Rate-limiting factors in dump leaching of pyritic ores. *Applied Mathematical Modelling*, 16:553–560, 1992.

G. Pantelis, A.I.M Ritchie, and Y.A. Stepanyants. A conceptual model for the description of oxidation and transport processes in sulphidic waste rock dumps. *Applied Mathematical Modelling*, 26:751–770, 2002.

J. Petersen. *Assessment and Modelling of Chromium Release in Minerals Processing Waste Deposits*. PhD thesis, University of Cape Town, South Africa, 1998.

J. Petersen. Personal communication, 2006.

J. Petersen and D.G. Dixon. Systematic modelling of heap leach process for optimization and design. In Patrick R. Taylor, D. Chandra, and R.G. Bautista, editors, *EPD Congress 2002*, pages 757–771, Pennsylvania,USA, 2002a. TMS.

J. Petersen and D.G. Dixon. Thermophilic heap leaching of a chalcopyrite concentrate. *Minerals Engineering*, 15:777–785, 2002b.

J. Petersen and D.G. Dixon. The dynamics of chalcocite heap bioleaching. In C.A. Young, A.M. Alfantazi, C.G. Anderson, D.B. Dreisinger, B. Harris, and A. James, editors, *Hydrometallurgy 2003*, volume 1, pages 351–364. TMS, 2003.

D.A. Ratkowsky, R.K. Lowry, T.A. McMeekin, A.N. Stokes, and R.E. Chandler. Model of bacterial culture growth rate throughout the entire biokinetic temperature range. *Journal of Bacteriology*, 154:1222–1226, 1983.

D.E. Rawlings, editor. *Biomining - Theory, Microbes and Industrial Processes*. Springer-Verlag, 1997a.

D.E. Rawlings. Mesophilic, autotrophic bioleaching bacteria: Description, physiology and role. Rawlings (1997a), chapter 11, pages 231–245.

D.E Rawlings. Heavy metal mining using microbes. *Annual Review of Microbiology*, 56: 65–91, 2002.

D.E Rawlings. Characteristics and adaptability of iron- and sulphur-oxidizing microorganisms used for the recovery of metals from minerals and their concentrates. *Microbial Cell Factories*, 4(13), 2005. [cited 7 October 2005]. Available from http://www.microbialcellfactories.com/content/4/1/13.

T. Rohwerder, T. Gehrke, K. Kinzler, and W. Sand. Bioleaching review part A: Progress in bioleaching: fundamentals and mechanisms of bacterial metal sulphide oxidation. *Applied Microbiology Biotechnology*, 63:239–248, 2003.

R.J. Roman and C. Olsen. Theoretical scale-up of heap leaching. In F.F. Aplan, W.A. McKinney, and A.D. Pernichele, editors, *Solution Mining Symposium*, pages 211–229, New York, 1974 AIME.

W. Sand and T. Gehrke. Extracellular polymeric substances mediate bioleaching/biocorrosion via interfacial processes involving iron(III) ions and acidophilic bacteria. *Research in Microbiology*, 157:49–56, 2006.

J.L. Shafer, M.L. White, and C.L. Caenepeel. Application of the shrinking core model for copper oxide leaching. *Mining Engineering*, 31:165–171, 1979.

G.A. Sheikhzadeh, M.A. Mehrabian, S.H. Mansouri, and A. Sarrafi. Computational modelling of unsaturated flow of liquid in heap leaching – using the results of column tests to calibrate model. *International Journal of Heat and Mass Transfer*, 48:279–292, 2005.

M. Sidborn, J.M. Casas, J. Martinez, and L. Moreno. Two dimensional dynamic model of a copper sulphide ore heap. *Hydrometallurgy*, 71:67–74, 2003.

M.P. Silverman and H.L. Ehrlich. Microbial formation and degradation of minerals. *Advances in Applied Microbiology*, pages 153–206, 1964.

Y. Tan, J. Gannon, P. Baveye, and M. Alexander. Transport of bacteria in an aquifer sand: Experiments and model simulations. *Water Resources Research*, 30(12):3243–3252, 1994.

H. Tributsch. Direct versus indirect leaching. In R. Amils and A. Ballester, editors, *Biohydrometallurgy and the Environment Toward the Mining of the 21st Century, Part A*, pages 51–60. Elsevier, 1999.

D.A. van Scherpenzeel, M. Boon, C. Ras, G.S. Hansford, and J.J. Heijnen. Kinetics of ferrous iron oxidation by *Leptospirillum* bacteria in continuous cultures. *Biotechnology progress*, 14:425 -433, 1998.

B. Volesky. Biosorption process simulation tools. *Hydrometallurgy*, 71:179–190, 2003.

M.H. Zwietering, J.T. De Koos, B.E. Hasenack, J.C. De Wit, and K. Van 'T Riet. Modelling of bacterial growth as a function of temperature. *Applied and Environmental Microbiology*, 57:1094–1101, 1991.

APPENDIX A

Model Implementation

A.1 Numerical Considerations

The consistency of the simulation results was checked by performing a mass balance at the end of each simulation. Total copper formed, based on the final conversion of the copper minerals, was compared to the total dissolved copper obtained during the simulation (that is, the sum of the amount of copper in the agglomerate at the end to the simulation, and the total copper that has been carried out of the system by the flowing phase). An iron balance was also performed: the total iron input into the system (total inflowing ferric and ferrous, plus the initial amount of ferric and ferrous in the agglomerate), was compared to the total iron output from the system (total outflowing ferric and ferrous, plus the final amount of ferric and ferrous at the end of the simulation).

Achieving a mass balance that was within reasonable error bounds proved to be the most challenging aspect of the numerical implementation. The point to note is that the system of equations is stiff. Roughly speaking, a system of differential equations is said to be stiff in a certain interval of integration if its numerical solution by some numerical methods require a step-size that is extremely small in relation to the smoothness of the exact solution in that interval in order to avoid instabilities. At the initial stage of the simulations, there are rapid variations in the reagent concentrations due to a steep diffusion gradient at the particle surfaces. Furthermore, the diffusion, reaction and advection mechanisms occur at different timescales. These phenomena contribute to the stiffness of the equations. The scheme that was used was to semi-discretize the partial differential equations using method of lines, and solve the resulting system of ordinary differential equations using a stiff ODE solver. The implemented scheme is explained in more detail in Sections A.2 and A.3. Using these methods, the maximum relative error in the mass balance calculations, for all the simulations that were run in this thesis, was $< 0.7\%$. Apart from the relative errors for the simulations involving high flow rates, all other simulations had relative errors $< 0.15\%$. Higher simulation errors were obtained

with increasing flowrates, the highest being about 0.65% for a flowrate of 2 cm/min.

Prior to the settling on the aforementioned scheme, a variety of numerical techniques were tried. This included backward Euler (implemented in semi-implicit form for the diffusion-reaction equation Eq.(3.1)), midpoint rule, Crank-Nicholson, and Runge-Kutta of orders 2 and 4. A main source of error was the finite difference implementation of the surface boundary condition of the ore particles. Initially, the following work-around was used:

1. Set the particle surface concentration of a species at a given time equal to the bulk concentration, and use this value to solve the diffusion-reaction equation, Eq.(3.1).
2. Calculate the contribution to the bulk concentration of the species, from a given size class, and for the present timestep, by taking the difference of the volume averaged concentration of the species in that particle size class before and after Eq.(3.1) was solved.
3. Calculate the rate of change of the bulk concentration by adding together the microbial source term, the rate of exchange with the flowing phase, and the weighted average of the total contribution from the particle size classes (weighed by the mass fraction of each size class) divided by time.

Expressing these steps, respectively, in mathematical terms:

1.

$$C_i(R_j, t) = C_{bi} \tag{A.1}$$

2.

$$S_{ps,ij} = \left(\frac{3D_i}{R_j}\right) \left.\frac{\partial C_i}{\partial r}\right|_{r_j = R_j} \tag{A.2}$$

$$\equiv \frac{3}{R_j^3} \int_0^{R_j} \left(\epsilon \frac{\partial C_i}{\partial t} - \sum_l \nu_{il} S_l\right) r^2 dr \tag{A.3}$$

The expression on the RHS of Eq.(A.3) is now discretized, using Simpson's rule, to

get:

$$S_{\mathrm{ps},ij} = \frac{3I}{\Delta t R_j^3} \quad \text{where,} \tag{A.4}$$

$$I = \frac{\Delta r}{3}(f_0 + f_N + 4\sum_{n\ \mathrm{odd}} f_n + 2\sum_{n\ \mathrm{even}} f_n); \quad (N \text{ even}) \text{ and} \tag{A.5}$$

$$f_n = \left(\epsilon(C_n^{(m+1)} - C_n^{(m)}) - \Delta t \sum_l \nu_{ll} S_{ln}^{(m)}\right)(n\Delta r)^2 \tag{A.6}$$

3.

$$\varepsilon_\mathrm{b}\frac{dC_{\mathrm{b}i}}{dt} = -3\epsilon\varepsilon_\mathrm{p}\sum_j w_j S_{\mathrm{ps},ij} + k_{\mathrm{bf}}^*(C_{\mathrm{f}i} - C_{\mathrm{b}i}) + \varepsilon_\mathrm{b}\sum_m \nu_{im} S_{\mathrm{bulk},m} \tag{A.7}$$

Although this method gave decent mass balance results, the flaw is in the assumption that the particle surface concentration of a species at any time is equal to the bulk concentration (that is, a Dirichlet boundary condition). A Neumann boundary condition (which is the current implementation) makes more sense; that is to say that flux across the ore particle boundary is driven by the concentration gradient at the particle surface.

The fact that the ODE solver that is presently being used does adaptive timestepping is a definite plus. The system experiences very rapid changes initially. Coupled with a finite difference discretization of the particle surface boundary conditions, this resulted not only in numerical inaccuracies, but sometimes in solutions that experienced initial oscillations. The ability to use very small timesteps initially, and relax this timestep later on when the solutions are slowly varying, is important.

It is possible that the (albeit minor) discrepancy that still exists in the mass balance can be resolved using a finite volume formulation for the space discretization. As previously stated, the finite difference discretization of the particle surface boundary conditions leads to some loss of mass. Finite volume methods have better mass conservation properties, and may therefore rectify this problem.

A.2 Space Discretization

The partial differential equations in the model were semi-discretized (in space), and the resulting system of ordinary differential equations was solved using an ODE solver. Eq.(3.1) (see page 29) was semi-discretized using second order central difference:

$$\frac{dC_n}{dt} = \frac{D}{(\Delta r)^2}\left[C_{n-1}\left(1-\frac{1}{n}\right) - 2C_n + C_{n+1}\left(1+\frac{1}{n}\right)\right] + \left(\sum_l \frac{\nu_{il}}{\epsilon} S_{ln}\right) + \left(\delta(R-r)\sum_m \frac{\nu_{im}}{\epsilon} S_{\mathrm{surf},mn}\right) . \quad \text{(A.8)}$$

The boundary conditions were implemented as follows:

- Discretizing

$$\left.\frac{\partial C}{\partial r}\right|_{r=0} = 0 , \quad \text{(A.9)}$$

we obtain

$$C_1 = C_{-1} . \quad \text{(A.10)}$$

Applying L'Hopital's rule to Eq.(3.1), we obtain,

$$\frac{\partial C}{\partial t} = 3D\frac{\partial^2 C}{\partial r^2} + \left(\sum_l \frac{\nu_{il}}{\epsilon} S_{ln}\right) + \left(\delta(R-r)\sum_m \frac{\nu_{im}}{\epsilon} S_{\mathrm{surf},mn}\right) . \quad \text{(A.11)}$$

Discretizing the first term on the RHS of Eq.(A.11) using central differences, and substituting Eq.(A.10) gives:

$$\frac{dC_0}{dt} = \frac{6D}{(\Delta r)^2}(C_1 - C_0) + \left(\sum_l \frac{\nu_{il}}{\epsilon} S_{l0}\right) + \left(\delta(R-r)\sum_m \frac{\nu_{im}}{\epsilon} S_{\mathrm{surf},m0}\right) . \quad \text{(A.12)}$$

- Discretizing

$$-D\left.\frac{\partial C}{\partial r}\right|_{r=R} = -k_{\mathrm{bs}}(C_b - C_R) \quad \text{(A.13)}$$

using central differences gives:

$$C_{N+1} = C_{N-1} + \frac{k_{\mathrm{bs}}}{a}(C_b - C_N) , \text{ where} \quad \text{(A.14)}$$

$$a = \frac{D}{2\Delta r} \text{ and } C_N = C_R \quad \text{(A.15)}$$

Hence the difference equation at $r = R$ is:

$$\frac{dC_N}{dt} = \frac{D}{(\Delta r)^2}\left[C_{N-1}\left(1-\frac{1}{N}\right) - 2C_N + \left(C_{N-1} + \frac{k_{\text{bs}}}{a}(C_b - C_N)\right)\left(1+\frac{1}{N}\right)\right] + \left(\sum_l \frac{\nu_{il}}{\epsilon} S_{lN}\right) + \left(\delta(R-r)\sum_m \frac{\nu_{im}}{\epsilon} S_{\text{surf},mN}\right) \quad \text{(A.16)}$$

Eq.(3.6) and Eq.(3.10) (see page 31) were semi-discretized using first order forward difference method:

$$\varepsilon_{\text{f}} \frac{dC_{\text{f}n}}{dt} = -u\left(\frac{C_{\text{f}n} - C_{\text{f}(n-1)}}{\Delta z}\right) - k^*_{\text{bf}}(C_{\text{f}n} - C_b) \; . \quad \text{(A.17)}$$

A.3 Time Integration

The time integration of the resulting system of ordinary differential equations was done using the odeint routine in the Scipy package.

Scipy is a collection of mathematical libraries for use with the Python programming language. The odeint routine is available in Scipy for integrating a first-order vector differential equation:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y}, t) \; , \quad \text{(A.18)}$$

with initial conditions $\mathbf{y}(0) = y_0$, where $\mathbf{y}$ is an N length vector. The routine is able to solve stiff and non-stiff systems. It provides implicit Adams method (up to order 12), for non-stiff problems, and a method based on backward differentiation formulas (up to order 5), for stiff problems. It is able to automatically choose a suitable method for a problem, and also to do adaptive stepsize control.

A.4 Program Overview

The codes were developed in Python (version 2.3.5) on a Linux workstation. Python is a powerful, cross-platform, object-oriented, scripting language that is freely available. An object-oriented approach was taken in the programming, and 5 classes were defined: the particle size distribution class, species class (for chemical species), microbes class, minerals class, and simpar class (to hold other simulation parameters that do not fit into the other

classes). These classes were stored in the classes.py module. The rest of the program was organized into the following modules: main.py (the main program, from which all other functions are called), preamble.py (contains code that is executed at the start of the program), engine.py (contains the integration, and other general purpose functions), IO.py (contains the input/output functions), and modulecheck.py (contains functions for checking that all required Python modules, and other external programs called by the code, are installed). Input parameters were defined in 5 input files: min_par (mineral parameters), species_par (chemical species parameters), psc_par (particle size distribution parameters), bact_par (microbial parameters) and sim_par (other simulation parameters).

The Python scripts and sample input files are listed in Appendix B. The program flowchart is shown in Fig.A.1.

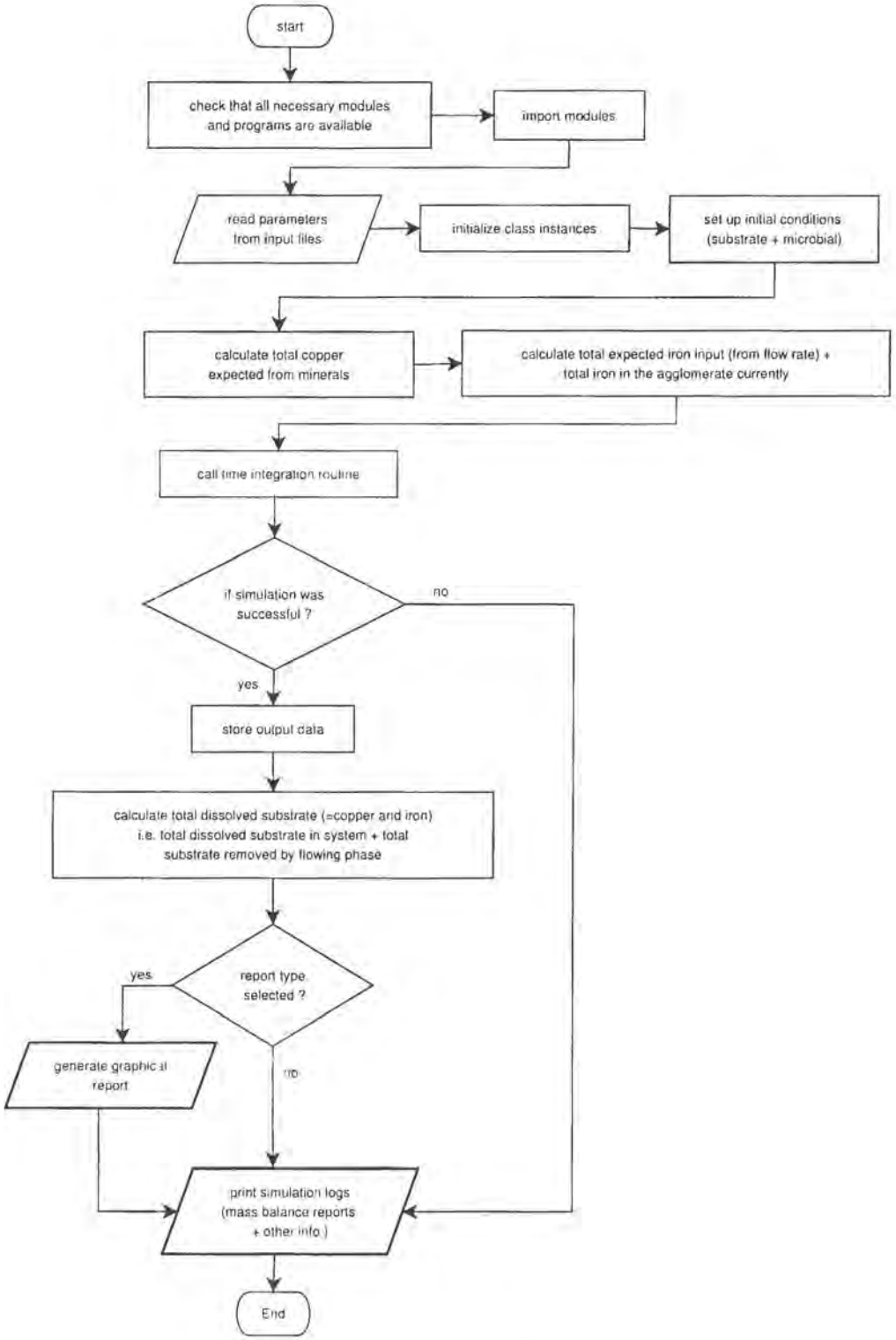


Figure A.1: Program flowchart.

APPENDIX B

Code Listing

B.1 Python Codes

The Python codes are formatted to allow for easier reading. For the Python novice, the color codes are explained as follows:

- keywords are in red and underlined.
- comments are in gray and italicized.
- strings are in purple and bold face.
- all other identifiers are in black.

Listing B.1: main.py

```
#!usr/bin/python
""" Main program
"""

from preamble import *
startTime=time.time()

#SETUP SIMULATION
#get the input directory
path=os.path.split(cwd)[0]
inputdir=os.path.join(path,'input-files')
#read data from input files
simpar,sizeclass_dict,minerals_dict,species_dict,microbes_dict=\
    initializeFromFiles('sim_par','psc_par','min_par','species_par','bact_par',\
            inputdir)

if simpar.microbes:
    text="Running Biotic leaching simulation..."
else:
    text="Running Abiotic leaching simulation..."
print text

#create output directory
simresults_dir=os.path.join(path,'simulation-results')
if not os.path.isdir(simresults_dir):
```

```
    os.mkdir(simresults_dir)
simpar.outdirname=os.path.join(simresults_dir,simpar.outdirname)
if os.path.isdir(simpar.outdirname):
        print "Output directory already exists. Overwriting ...\n"
        shutil.rmtree(simpar.outdirname)
os.mkdir(simpar.outdirname)

#RUN SIMULATION
simstart=time.time()
log=simulate(simpar,sizeclass_dict,minerals_dict,microbes_dict,species_dict)
simstop=time.time()
text2= 'Simulation time=%g hrs' % (simpar.tstop/60)
print text2
print log
text+='\n'+text2
log=string.join([text,log],'\n')
#more logs
simpar.getLogs()

#STORE OUTPUT DATA IN FILES
#create output directories--contd
data_outdir =os.path.join(simpar.outdirname,'data')
os.mkdir(data_outdir)
report_outdir=os.path.join(simpar.outdirname,'graphs')
os.mkdir(report_outdir)
#create output data files
createDataFiles(simpar,minerals_dict.keys(),species_dict.keys(),\
    microbes_dict.keys(),sizeclass_dict.keys(),data_outdir)
datafiles=simpar.outfiles['files3d']
#write data
for sc in sizeclass_dict.keys():
    for sname in species_dict.keys():
        writeData3d(datafiles[sc][sname],sizeclass_dict[sc].r,\
            simpar.time_list,species_dict[sname].psolutions[sc])
    for mn in minerals_dict.keys():
        writeData3d(datafiles[sc][mn],sizeclass_dict[sc].r,\
            simpar.time_list,minerals_dict[mn].solutions[sc])
    if simpar.pmicrobes:
        for micname in microbes_dict.keys():
            writeData3d(datafiles[sc][micname],sizeclass_dict[sc].r,\
                simpar.time_list,microbes_dict[micname].psolutions[sc])
writeData2d(simpar.outfiles['files2d'],simpar.time_list,species_dict,\
    microbes_dict,minerals_dict)
if simpar.microbes:
    writeMisc(simpar,species_dict['ferrous'],microbes_dict)
#close data files
closeDataFiles(simpar.outfiles)

#GENERATE REPORTS
#create reports-- is done only if integration was successful
#resolve report format
if simpar.report_style=='eps':
    if rstyle['eps']:
```

```
            fileType='eps'
        elif rstyle['png']:
            print "***Ghostscript is not installed(or not in PATH). \
                Generating html report"
            fileType='png'
    elif simpar.report_style=='postscript':
        if rstyle['eps']:
            fileType='eps'
            if not rstyle['ps']:
                print "***epsmerge not installed. Generating eps report"
        elif rstyle['png']:
            print "***Ghostscript is not installed (or not in PATH). \
                Generating html report"
            fileType='png'
    elif simpar.report_style=='html':
        fileType='png'
    else:
        simpar.plot2d=False; simpar.plot3d=False
        print "***Invalid report style. No graphical output will be generated.\n"
    #create reports
    if not (simpar.plot2d == False and simpar.plot3d == False):
        files2d={};files3d=[]
        if simpar.plot2d:
            files2d['bulkConc']=plot2d(simpar.outfiles['files2d']['bulkConc'],\
                'bulk',species_dict.keys(),microbes_dict.keys(),fileType,\
                report_outdir);
            files2d['bulkConc'].sort()
            files2d['flowConc']=plot2d(simpar.outfiles['files2d']['flowConc'],\
                'flow',species_dict.keys(),microbes_dict.keys(),fileType,\
                report_outdir);
            files2d['flowConc'].sort()
            files2d['avePsolns']= plot2dAve(simpar.outfiles['files2d']['avePsolns'],\
                sizeclass_dict,species_dict.keys(),minerals_dict.keys(),fileType,
                  report_outdir)
            if simpar.microbes:
                files2d['surfaceConc']=plotSurf(\
                    simpar.outfiles['files2d']['surfaceConc'],sizeclass_dict,\
                        species_dict.keys(),fileType,report_outdir)
        if simpar.plot3d:
                files3d=plot3d(simpar.plot_gap,datafiles,species_dict.keys(),\
                    microbes_dict.keys(),sizeclass_dict,fileType,report_outdir);
                files3d.sort()
        if fileType=='eps':
            if simpar.report_style=='postscript' and rstyle['ps']:
                filename2d='bulkConc.ps,flowConc.ps,AvParticleConc.ps,SurfaceConc.ps'
                filename3d='plots3d.ps'
                psReport(filename2d,filename3d,files2d,files3d,simpar)
        elif fileType=='png':
            filename='results.html'
            htmlReport(filename,files2d,files3d,simpar)
    #copy input dir to output directory for reference and
    #to keep track of input values used for a given run
    outindir=os.path.join(simpar.outdirname,'input_files')
```

```
import shutil
shutil.copytree(inputdir,outindir)

#END SIMULATION
stopTime=time.time()
simTime=(simstop-simstart)/60
totalTime=(stopTime-startTime)/60 #almost!
text="Solving time = %.5f mins\nTotal time = %.5f mins\n " % (simTime,totalTime)
print text
log +=text
log_file=simpar.outfiles['misc']['simlog']
log_file.write(log)
log_file.write(simpar.log)
log_file.close()
print "All outputs are in %s\n" % simpar.outdirname
```

Listing B.2: engine.py

```
#!/usr/bin/python
"""General purpose functions

This module contains general purpose functions used in the program
"""
import sys
from scipy.integrate import simps,odeint
from Numeric import ones,zeros,array,ceil,matrixmultiply,arange
import os

def errorHandler(message):
    message = "***Error: " +message
    print message
    print "*****program terminated prematurely*****"
    sys.exit()
#-----------------------------------------------------------------------------
def tryCommand(command):
    failure=command
    if failure:
        message="***Running command (%s) failed" % str(command)
        errorHandler(message)
#-----------------------------------------------------------------------------
def storeSolutions(solnarray,tindx,simpar,sizeclass,minerals,species,microbes):
    """Store simulation solutions into the respective classes.
    """
    skeys=species.keys()
    if 'ferric' in skeys : skeys.remove('ferric')
    if 'ferrous' in skeys : skeys.remove('ferrous')
    minNames=minerals.keys();
    if 'sulfur' in minNames: minNames.remove('sulfur')
    #minerals
    for mn in minNames:
        for sc in sizeclass.keys():
            minerals[mn].solutions[sc].append(\
                solnarray[tindx][minerals[mn].conv_indx[sc][0]:minerals[mn].\
```

```
                conv_indx[sc][1]]*100)
    #sulfur grade
    sulfur=minerals['sulfur']
    for sc in sizeclass.keys():
        sulfur.solutions[sc].append(solnarray[tindx][sulfur.grade_indx[sc][0]:\
            sulfur.grade_indx[sc][1]]) #g/g-ore
    #chemical species
        #ferric/ferrous
    for sp in ['ferric','ferrous']:
        species[sp].bulk_solutions.append(solnarray[tindx][species[sp].bulk_indx]\
            *species[sp].mmass)
        species[sp].flow_solutions.append(solnarray[tindx][species[sp].flow_indx]*\
            species[sp].mmass)
        for sc in sizeclass.keys():
            massconc=solnarray[tindx][species[sp].pconc_indx[sc][0]:species[sp].\
                pconc_indx[sc][1]]*species[sp].mmass*simpar.porosity #(g/cm^3_p) *1e-3
            species[sp].psolutions[sc].append(massconc)
            species[sp].surface_solutions[sc].append(solnarray[tindx][species[sp].\
                pconc_indx[sc][1]-1]*species[sp].mmass*simpar.porosity)#(g/cm^3_p)*1e-3
            #range for ferric/ferrous graphs
            sizeclass[sc].minFezrange=min(sizeclass[sc].minFezrange,min(massconc))
            sizeclass[sc].maxFezrange=max(sizeclass[sc].maxFezrange,max(massconc))
    #other chemical species
    for sp in skeys:
        species[sp].bulk_solutions.append(solnarray[tindx][species[sp].bulk_indx]*\
            species[sp].mmass) #g/L
        species[sp].flow_solutions.append(solnarray[tindx][species[sp].flow_indx]*\
            species[sp].mmass) #g/L
        for sc in sizeclass.keys():
            massconc=solnarray[tindx][species[sp].pconc_indx[sc][0]:species[sp].\
                pconc_indx[sc][1]]*species[sp].mmass*simpar.porosity
            species[sp].psolutions[sc].append(massconc)
            species[sp].surface_solutions[sc].append(solnarray[tindx][species[sp].\
                pconc_indx[sc][1]-1]*species[sp].mmass*simpar.porosity)
    #microbial species
    for mc in microbes.keys():
        #unitConv converts cells/g_ore->cells/Lb
        unitConv = simpar.ore_density*(1-simpar.porosity)*simpar.solid_volfrac*\
            1000/simpar.bulk_volfrac
        ads_pop_mc = solnarray[tindx][microbes[mc].ads_pop_indx]*unitConv
        des_pop_mc = solnarray[tindx][microbes[mc].des_pop_indx]
        if ads_pop_mc < 0:
            ads_pop_mc = 0.0
        bulk_pop=ads_pop_mc + des_pop_mc
        microbes[mc].bulk_pop_solutions.append(bulk_pop/((1e10))) #cells/Lbulk
        microbes[mc].ads_solutions.append(ads_pop_mc/(1e10)) #cells/Lbulk
        microbes[mc].des_solutions.append(des_pop_mc/(1e10)) #cells/Lbulk
        microbes[mc].flow_solutions.append(solnarray[tindx][microbes[mc].\
            flow_pop_indx]/(1e10)) #cells/Lflow
    #store
    if simpar.microbes:
        for mc in microbes.keys():
            for sc in sizeclass.keys():
```

```
                microbes[mc].agr_solutions[sc].append(microbes[mc].\
                    agr_solutions2[sc][tindx])
            microbes[mc].dgr_solutions.append( microbes[mc].dgr_solutions2[tindx])
        for sc in sizeclass.keys():
            species['ferrous'].sr_solutions[sc].append(species['ferrous'].\
                sr_solutions2[sc][tindx])
        species['ferrous'].br_solutions.append(species['ferrous'].\
            br_solutions2[tindx])
#-------------------------------------------------------------------------------
def storeAverageParticleSolutions(sizeclass,species,minerals):
    """Calculate and store average solutions for particle size classes.
    """
    for sc in sizeclass.keys():
        for mn in minerals.keys():
            #get and store average
            int_fac =minerals[mn].solutions[sc]*\
                sizeclass[sc].r**2
            minerals[mn].average_solutions[sc].extend(3*\
                simps(int_fac,sizeclass[sc].r)/sizeclass[sc].R**3)
        for sp in species.keys():
            #get and store average
            int_fac =species[sp].psolutions[sc]*sizeclass[sc].r**2
            species[sp].average_psolutions[sc].extend(\
                3*simps(int_fac,sizeclass[sc].r)/sizeclass[sc].R**3)
#-------------------------------------------------------------------------------
def xdotAbiotic(x,t,simpar,sizeclass,minerals,species):
    """Derivative function for abiotic simulation.

    Returns the rhs vector f(y,t) for the system of equations:
    dy/dt = f(y,t);
    """
    xdot = zeros((len(x),),'d')
    minkeys=minerals.keys()
    if simpar.sulfur_present: minkeys.remove('sulfur')
    #calculate mineral conversion rates
    for mn in minkeys:
        minerals[mn].updateRxnRate(x,simpar,sizeclass,species['ferric'],\
            species['ferrous'],species['acid']) #mol/cm^3p-min
    #acid/gangue rate
    getAcidGangueRate(x,species['acid'],simpar,sizeclass)

    #setup differentiation vector xdot
    #--minerals
    for sc in sizeclass.keys():
        sulfur_rate=zeros((sizeclass[sc].N,),'d')
        for mn in minkeys:
            minscInd=minerals[mn].conv_indx[sc]
            xdot[minscInd[0]:minscInd[1]] = minerals[mn].rate[sc]/(\
                simpar.ore_density*(1-simpar.porosity)*minerals[mn].grade[sc])
             #calculate the rate of change of sulfur grade due the the reaction of
                mineral 'mn'
            if minerals[mn].type == 'sulfide':
                sulfur_rate += minerals[mn].rate[sc]*minerals[mn].Sstoic*\
```

```
                    minerals['sulfur'].mmass #->g/cm^3p_sc-min
                #net rate of change of sulfur grade at the particle surface
                    #is the rate of formation from the sulfide minerals
            if simpar.sulfur_present:
                sulfur_rate/=(simpar.ore_density*(1-simpar.porosity)) #g/g_ore-min
                xdot[ minerals['sulfur'].grade_indx[sc][0]:minerals['sulfur'].\
                    grade_indx[sc][1] ] = sulfur_rate
        #--species
        for sp in species.keys():
            exchbs =0 #exchange at bulk/surface interface
            bconc = x[species[sp].bulk_indx]
            fconc = x[species[sp].flow_indx]
            for sc in sizeclass.keys():
                #get the source term
                minSource = species[sp].getSource(sp,sc,simpar,minerals,\
                    species['acid'].rate) #mol/cm^3_p
                spscInd = species[sp].pconc_indx[sc]
                pconc = x[spscInd[0]:spscInd[1]]
                factor = species[sp].diff/(2*sizeclass[sc].dr)
                xpend = (species[sp].k/factor)* bconc * (1+1./(sizeclass[sc].N-1))*\
                    (species[sp].diff/sizeclass[sc].dr**2) #factor to add at r=R -- mass
                        transfer bulk/surface
                xdot[spscInd[0]:spscInd[1]] = matrixmultiply(\
                    species[sp].diffMatrix[sc],pconc)
                xdot[spscInd[1]-1] += xpend
                xdot[spscInd[0]:spscInd[1]] += minSource*1000/simpar.porosity
                exchbs += (sizeclass[sc].massfrac/sizeclass[sc].R) * \
                    (bconc - x[spscInd[1]-1])
            exchbf = simpar.kbf * (fconc-bconc) #exchange at the bulk/flowing interface
            #derivative for the stagnant and flowing phases
            if simpar.flow_volfrac > 0:
                xdot[species[sp].bulk_indx] = -3*simpar.porosity*simpar.solid_volfrac*\
                    species[sp].k*exchbs + exchbf
                xdot[species[sp].bulk_indx] /= simpar.bulk_volfrac
                xdot[species[sp].flow_indx] = (simpar.uf/simpar.L) * \
                    (species[sp].inletflow_conc - fconc) - exchbf
                xdot[species[sp].flow_indx] /= simpar.flow_volfrac
            else:
                xdot[species[sp].bulk_indx] = -3*simpar.porosity*simpar.solid_volfrac*\
                    species[sp].k*exchbs
                xdot[species[sp].bulk_indx] /= simpar.bulk_volfrac
                xdot[species[sp].flow_indx] = 0
        return xdot
    #-----------------------------------------------------------------------------
    def xdotBiotic(x,t,simpar,sizeclass,minerals,species,microbes):
        """Derivative function for biotic simulation.

        Returns the rhs vector f(y,t) for the system of equations:
        dy/dt = f(y,t);
        """
        xdot = zeros((len(x),),'d')
        minkeys=minerals.keys()
        if simpar.sulfur_present: minkeys.remove('sulfur')
```

```
        #calculate source terms
        #--calculate microbial growth rates
        for mc in microbes.keys():
            microbes[mc].updateGrowthRate(x,simpar,sizeclass,\
                species,minerals['sulfur'])
        #--calculate mineral conversion rates
        for mn in minkeys:
200         minerals[mn].updateRxnRate(x,simpar,sizeclass,\
                species['ferric'],species['ferrous'],species['acid']) #mol/cm^3_p
        #--calculate other rate terms
        getMicrobialSurfaceRate(species['ferrous'],x,simpar,\
            sizeclass,microbes,'iron')
        getMicrobialBulkRate(species['ferrous'],simpar,microbes)
        getMicrobialSurfaceRate(minerals['sulfur'],x,simpar,\
            sizeclass,microbes,'sulfur')
        getAcidGangueRate(x,species['acid'],simpar,sizeclass)

210     #setup source vector xdot
        #--minerals
        for sc in sizeclass.keys():
            sulfur_rate=zeros((sizeclass[sc].N,),'d')
            for mn in minkeys:
                minscInd = minerals[mn].conv_indx[sc]
                xdot[ minscInd[0]:minscInd[1] ] =\
                    minerals[mn].rate[sc]/((1-simpar.porosity)*\
                        simpar.ore_density*minerals[mn].grade[sc])
                #calculate the rate of change of sulfur grade due the the reaction of
                  mineral 'mn'
220             if minerals[mn].type == 'sulfide':
                    sulfur_rate += minerals[mn].rate[sc]*\
                        minerals[mn].Sstoic*minerals['sulfur'].mmass #->g/cm^3p_sc-min
                    #net rate of change of sulfur grade at the particle surface is
                    #the rate of formation from the sulfide minerals minus the rate
                    #of consumption by the sulfur oxidizing microbes
                    sulfur_rate[-1] -= minerals['sulfur'].surface_rate[sc]*\
                        minerals['sulfur'].mmass #g/cm^3p_sc-min
            if simpar.sulfur_present:
                sulfur_rate/=(simpar.ore_density*(1-simpar.porosity)) #g/g_ore-min
230             xdot[ minerals['sulfur'].grade_indx[sc][0]:minerals['sulfur'].\
                    grade_indx[sc][1] ] = sulfur_rate
        #--microbes
        for mc in microbes.keys():
            #adspop rate
            avadspopgr=0 #average adsorbed population growth rate
            for sc in sizeclass.keys():
                avadspopgr += sizeclass[sc].massfrac * microbes[mc].ads_growth_rate[sc]
            #apply the Langmuir isotherm
            ads_pop = (microbes[mc].ads_max * microbes[mc].adsEq_const*x[microbes[mc].\
240             des_pop_indx])/(1+microbes[mc].adsEq_const*x[microbes[mc].des_pop_indx])
            xdot[microbes[mc].ads_pop_indx] = ads_pop * (avadspopgr - \
                microbes[mc].death_rate)
            #despop rate
            dpIndx=microbes[mc].des_pop_indx
```

```
        fpIndx=microbes[mc].flow_pop_indx
        exchbfm = simpar.kbf*(x[fpIndx] - x[dpIndx])
        if simpar.flow_volfrac>0:
            xdot[dpIndx] =x[microbes[mc].des_pop_indx]*(\
                microbes[mc].des_growth_rate - microbes[mc].death_rate)+\
                    exchbfm/simpar.bulk_volfrac
            xdot[fpIndx] = (simpar.uf/simpar.L)*\
                (microbes[mc].inlet_flow_pop-x[fpIndx]) - exchbfm
            xdot[fpIndx] /=simpar.flow_volfrac
        else:
            xdot[dpIndx] =x[microbes[mc].des_pop_indx]*\
                (microbes[mc].des_growth_rate - microbes[mc].death_rate)
            xdot[fpIndx] = 0
    #--chemical species
    for sp in species.keys():
        bconc = x[species[sp].bulk_indx] #conc in bulk phase
        fconc = x[species[sp].flow_indx] #conc in flowing phase
        #surface rates
        fe2srate=species['ferrous'].surface_rate
        Ssrate =minerals['sulfur'].surface_rate
        exchbs=0 #exchange at bulk/surface interface
        for sc in sizeclass.keys():
            spscInd=species[sp].pconc_indx[sc]#indicies
            minSource = species[sp].getSource(sp,sc,simpar,\
                minerals,species['acid'].rate,microbes) #mol/cm^3_p
            surfSource= species[sp].getSurfaceSource(sp,simpar,\
                fe2srate[sc],Ssrate[sc]) #mol/cm^3p_sc-min
            pconc = x[spscInd[0]:spscInd[1]] #conc in particle
            factor= species[sp].diff/(2*sizeclass[sc].dr)
            xpend = (species[sp].k/factor)*bconc*(1+1./(sizeclass[sc].N-1))*\
                (species[sp].diff/sizeclass[sc].dr**2) #factor to add at r=R -- mass
                    transfer bulk/surface
            xdot[spscInd[0]:spscInd[1]] = matrixmultiply(\
                species[sp].diffMatrix[sc],pconc)
            xdot[spscInd[1]-1] +=xpend
            #add source term
            xdot[spscInd[0]:spscInd[1]] +=minSource*1000/simpar.porosity
            #additional source term at particle surface due to microbial reactions
            xdot[spscInd[1]-1] += surfSource*1000/simpar.porosity
            exchbs += (sizeclass[sc].massfrac/sizeclass[sc].R) *\
                (bconc-x[spscInd[1]-1])
        exchbf = simpar.kbf*(fconc-bconc) #exchange at bulk/flowing interface
        bulkSource =simpar.bulk_volfrac*simpar.bstoic[sp][0]*\
            species['ferrous'].bulk_rate #only ferrous has a bulk oxidation rate
        if simpar.flow_volfrac>0:
            if sp in simpar.bulkspeciesNames:
                xdot[species[sp].bulk_indx]= -3*simpar.porosity*\
                    simpar.solid_volfrac*species[sp].k*exchbs + bulkSource + exchbf
                xdot[species[sp].bulk_indx]/=simpar.bulk_volfrac
            else:
                xdot[species[sp].bulk_indx]=0
            xdot[species[sp].flow_indx]=(simpar.uf/simpar.L)*\
                (species[sp].inletflow_conc-fconc) - exchbf
```

```
                xdot[species[sp].flow_indx]/=simpar.flow_volfrac
            else:
                if sp in simpar.bulkspeciesNames:
                    xdot[species[sp].bulk_indx]= -3*simpar.porosity*\
                        simpar.solid_volfrac*species[sp].k*exchbs + bulkSource
                    xdot[species[sp].bulk_indx]/=simpar.bulk_volfrac
                else:
                    xdot[species[sp].bulk_indx]=0
                xdot[species[sp].flow_indx]=0
        #store the microbial oxidation rates of ferrous at the psc
        #surfaces and in the bulk_solutions. Also store the microbial growth rates.
        for mc in microbes.keys():
            for sc in sizeclass.keys():
                microbes[mc].agr_solutions[sc].append(\
                    microbes[mc].ads_growth_rate[sc])
            microbes[mc].dgr_solutions.append(microbes[mc].des_growth_rate)
        for sc in sizeclass.keys():
            species['ferrous'].sr_solutions[sc].append(\
                species['ferrous'].surface_rate[sc])
        species['ferrous'].br_solutions.append(species['ferrous'].bulk_rate)
        return xdot
    #-----------------------------------------------------------------------------
    def simulate(simpar,sizeclass,minerals,microbes,species):
        """Main simulation routine.

            The tasks undertaken in this routine are (in chronological order):
            1. Do initial mass balances.
            2. Set up the initial input vector.
            3. Call the integration routine.
            4. Store the simulation output to the respective classes.
            5. Calculate the effective copper conversion.
            6. Do final mass balances.
        """
        #define extra structures for ferrous species
        if simpar.microbes:
            species['ferrous'].surface_rate={}; species['ferrous'].bulk_rate=0
            species['ferrous'].sr_solutions={}; species['ferrous'].br_solutions=[];
            for sc in sizeclass.keys():
                species['ferrous'].surface_rate[sc]=0
                species['ferrous'].sr_solutions[sc]=[]

        #species with non-constant concentration in the bulk
        simpar.bulkspeciesNames=[]
        for sp in species.keys():
            if not simpar.constant_in_bulk[sp]: simpar.bulkspeciesNames.append(sp)

        #do initial balance
        #--calculate total expected copper
        getExpectedCopper(species['copper'],simpar,sizeclass,minerals)
        #--calculate total iron input
        getExpectedIron(simpar,sizeclass,species['ferric'],species['ferrous'])

        #setup the initial input vector; order:minerals,species,microbes
```

```
    x=[]; indx=0;
    #minerals
    minkeys=minerals.keys()
    if 'sulfur' in minkeys: minkeys.remove('sulfur')
    for mn in minkeys:
        for sc in sizeclass.keys():
            minerals[mn].conv_indx[sc].append(indx)
            x.extend(zeros((sizeclass[sc].N,),'d')) #initial conversion of minerals is
                zero.
            indx += sizeclass[sc].N
            minerals[mn].conv_indx[sc].append(indx)
    if simpar.sulfur_present:
        for sc in sizeclass.keys():
            minerals['sulfur'].grade_indx[sc].append(indx)
            x.extend(ones((sizeclass[sc].N,),'d')*0.00) #initial sulfur formed from
                sulfide minerals is zero.
            indx += sizeclass[sc].N
            minerals['sulfur'].grade_indx[sc].append(indx)
    #species
    for sp in species.keys():
        for sc in sizeclass.keys():
            species[sp].pconc_indx[sc].append(indx)
            x.extend(ones((sizeclass[sc].N,),'d')*species[sp].pconc) #uniform species
                concentration. Specified by user in input file and the same for all
                sizeclasses.
            indx += sizeclass[sc].N
            species[sp].pconc_indx[sc].append(indx)
        x.extend([species[sp].bulk_conc,species[sp].flow_conc])
        species[sp].bulk_indx = indx; indx +=1
        species[sp].flow_indx = indx; indx +=1
    #microbes
    if simpar.microbes:
        for mc in microbes.keys():
            if simpar.pmicrobes:
                for sc in sizeclass.keys():
                    microbes[mc].pconc_indx[sc].append(indx)
                    x.extend(zeros((sizeclass[sc].N,),'d')) #initial microbial conc.
                        inside particles
                    indx += sizeclass[sc].N
                    microbes[mc].pconc_indx[sc].append(indx)
            x.extend([microbes[mc].ads_pop,microbes[mc].des_pop,\
                microbes[mc].flow_pop]);
            microbes[mc].ads_pop_indx = indx; indx+=1;
            microbes[mc].des_pop_indx = indx; indx+=1;
            microbes[mc].flow_pop_indx = indx; indx+=1;
    x=array(x,'d')

    #call integration routine
    if simpar.microbes:
        if simpar.pmicrobes:
            pass
        else:
            soln,simpar.infodict = odeint(xdotBiotic,x,simpar.time_list,\
```

```
            args=(simpar,sizeclass,minerals,species,microbes),full_output=1)
    else:
        soln,simpar.infodict = odeint(xdotAbiotic,x,simpar.time_list,\
            args=(simpar,sizeclass,minerals,species),full_output=1)
    soln = array(soln,'d')
    simpar.soln=soln ###

    #what to do if simulation fails:
    if not 'Integration successful' in simpar.infodict['message']:
        filename=os.path.join(simpar.outdirname,'simlogs.log')
        logfile =open(filename,'w')
        simpar.getLogs()
        logfile.write(simpar.log)
        logfile.close()
        message = "Simulation logs in %s\n" % simpar.outdirname
        errorHandler(message)

    #store microbial growth rates
    if simpar.microbes:
        for mc in microbes.keys():
            microbes[mc].agr_solutions2={}
            for sc in sizeclass.keys():
                temp=microbes[mc].agr_solutions[sc][:]
                microbes[mc].agr_solutions[sc]=[]
                microbes[mc].agr_solutions2[sc]=[]
                microbes[mc].agr_solutions2[sc][:] = \
                    [temp[tt-1] for tt in simpar.infodict['nfe']]
                microbes[mc].agr_solutions2[sc].insert(0,temp[0])
            temp=microbes[mc].dgr_solutions[:]
            microbes[mc].dgr_solutions=[];microbes[mc].dgr_solutions2=[]
            microbes[mc].dgr_solutions2[:]=\
                [temp[tt-1] for tt in simpar.infodict['nfe']]
            microbes[mc].dgr_solutions2.insert(0,temp[0])
        #store microbial ferrous oxidation rates
        species['ferrous'].sr_solutions2={}
        for sc in sizeclass.keys():
            temp=species['ferrous'].sr_solutions[sc][:]
            species['ferrous'].sr_solutions[sc]=[]
            species['ferrous'].sr_solutions2[sc]=[]
            species['ferrous'].sr_solutions2[sc][:] =\
                [temp[tt-1] for tt in simpar.infodict['nfe']]
            species['ferrous'].sr_solutions2[sc].insert(0,temp[0])
        temp=species['ferrous'].br_solutions[:]
        species['ferrous'].br_solutions=[]
        species['ferrous'].br_solutions2=[]
        species['ferrous'].br_solutions2[:]=\
            [temp[tt-1] for tt in simpar.infodict['nfe']]
        species['ferrous'].br_solutions2.insert(0,temp[0])

    #copy simulation output to respective classes
    for tt in range(int(simpar.timesteps)):
        storeSolutions(soln,tt,simpar,sizeclass,minerals,species,microbes)
    storeAverageParticleSolutions(sizeclass,species,minerals)
```

```
    #do final mass balance.
    #--calculate total copper formed
    log=copperBalance(simpar,sizeclass,minerals,species['copper'])
    #--calculate total iron output
    log+=ironBalance(simpar,sizeclass,species['ferric'],species['ferrous'])

    #calculate effective copper conversion
    effCopperConv(species['copper'],simpar,sizeclass,minerals)

    #return mass balance logs
    return log
#-------------------------------------------------------------------------------
def copperBalance(simpar,sizeclass,minerals,copper):
    """Do a copper balance and return formatted output
    """
    minNames=minerals.keys()
    #remove key sulfur from minerals keys.
    if 'sulfur' in minNames: minNames.remove('sulfur')

    cpm =[simpar.pstoic[mn][0] for mn in minNames] #moles of copper formed per mole of
       mineral reacted

    #copper formed from minerals
    copper_formed={} #au copper formed by each mineral
    tot_copper_formed=0 #copper from minerals
    for cf,name in zip(cpm,minNames):
        copper_formed[name]=0
        for sc in sizeclass.keys():
            intfac=(minerals[name].solutions[sc][-1]*sizeclass[sc].r**2)/100
            avConv=3*simps(intfac,sizeclass[sc].r)/sizeclass[sc].R**3
            minReacted=minerals[name].grade[sc]*avConv*\
                simpar.ore_density*(1-simpar.porosity) #mol/cm^3-p
            copper_formed[name]+=cf*minReacted*sizeclass[sc].massfrac
        tot_copper_formed+=copper_formed[name]
    tot_copper_formed*=simpar.solid_volfrac*copper.mmass*simpar.L**3 #g
    #copper concentration in particle, bulk and flowing phase
    pcopper,bcopper,fcopper=aveAggConc(simpar,sizeclass,copper) #g
    #calculate average amount of copper that left the system
    copper_outflow=simpar.soln[:,copper.flow_indx]*copper.mmass*\
        simpar.uf*simpar.L**2/1000
    copper_outflow = simps(copper_outflow,simpar.time_list) #g

    #print results
    result="Copper mass balance results (mol/cm^3-p)\n"
    for name,cf in zip(minNames,cpm):
        if cf:
            result+=name + "\n"
            result+=" Copper expected = %g\n" % copper.expN[name]
            result+=" Copper formed = %g = %.5f" % (copper_formed[name],\
                (copper_formed[name]*100/copper.expN[name])) + "%\n"
    result+='-'*52+'\n'
    result+="Copper mass balance results (g)\n"
```

```
    result+="Total copper expected = %.9f\n" % copper.expected
    result+="Copper formed so far as calculated from :\n"
    result+="(*) Minerals = %.9f = %.5f" % \
        (tot_copper_formed,(tot_copper_formed*100/copper.expected)) +'%\n'
    result+="(*) Dissolved copper = %.9f = %.5f" % \
        (((bcopper+pcopper+fcopper+copper_outflow)),\
            ((bcopper+pcopper+fcopper+copper_outflow)*100/copper.expected)) +'%\n'
    result+=" Copper in particles = %.9f = %.5f" % \
        (pcopper,(pcopper*100/copper.expected)) +'%\n'
    result+=" Copper in bulk = %.9f = %.5f" % \
        (bcopper,(bcopper*100/copper.expected)) +'%\n'
    result+=" Copper in flowing = %.9f = %.5f" %
        (fcopper,(fcopper*100/copper.expected)) +'%\n'
    result+=" Total copper outflow = %.9f = %.5f" % \
        (copper_outflow,(copper_outflow*100/copper.expected)) +'%\n'
    result+='.'*52+'\n'

    return result
#-------------------------------------------------------------------------------
def ironBalance(simpar,sizeclass,ferric,ferrous):
    """Do an iron balance and return formatted output
    """
    pferrous,aferrous,fferrous=aveAggConc(simpar,sizeclass,ferrous)
    pferric,aferric,fferric=aveAggConc(simpar,sizeclass,ferric)

    ferrous_outflow=simpar.soln[:,ferrous.flow_indx]*ferrous.mmass*\
        simpar.uf*simpar.L**2/1000 #g
    ferrous_outflow = simps(ferrous_outflow,simpar.time_list)
    ferric_outflow=simpar.soln[:,ferric.flow_indx]*ferrous.mmass*\
        simpar.uf*simpar.L**2/1000 #g
    ferric_outflow = simps(ferric_outflow,simpar.time_list)

    simpar.tot_iron_output=ferrous_outflow+ferric_outflow
    tot_iron_in_system =pferrous+aferrous+pferric+aferric+fferrous+fferric

    result="Iron mass balance results (g)\n"
    if simpar.flow_volfrac>0:
        result+="Total iron input = %.5f\n" % \
            (simpar.init_total_iron+simpar.tot_iron_input)
        result+="Final iron = %.5f\n" % (tot_iron_in_system+simpar.tot_iron_output)
        result+=" Final iron in system = %.5f\n" % tot_iron_in_system
        result+=" Total iron outflow = %.5f\n" % simpar.tot_iron_output
    else:
        result+="Total iron input = %.5f\n" % (simpar.init_total_iron)
        result+="Final iron in system = %.5f\n" % tot_iron_in_system

    result+='.'*52+'\n'
    return result
#-------------------------------------------------------------------------------
def effCopperConv(copper,simpar,sizeclass,minerals):
    """Calculate the effective copper conversion.

    effective copper conversion=(average copper formed)/(expected copper)
```

```
    """
    minNames=minerals.keys(); psc=sizeclass.keys()
    #remove key sulfur from minerals keys.
    if 'sulfur' in minNames: minNames.remove('sulfur')

    cpm=[simpar.pstoic[mn][0] for mn in minNames] #moles of copper formed per mole of
       mineral reacted
    #get info only for minerals that form copper
    for cf,mn in zip(cpm,minNames):
        if cf==0:
            del cf; del mn
    #get length of array
    length=len(minerals[minNames[0]].solutions[psc[0]])
    copper.eff_conv=zeros((length,),'d')

    for i in xrange(length): #for each timelevel
        for cf,mn in zip(cpm,minNames):
            avsum=0
            for sc in psc:
                intfac=minerals[mn].solutions[sc][i]*sizeclass[sc].r**2
                average=(3*simps(intfac,sizeclass[sc].r)/sizeclass[sc].R**3)/100 #
                   conversion from percentage
                #weigh by sizeclass
                avsum+=sizeclass[sc].massfrac*average*minerals[mn].grade[sc] #mol/g_o
            copper.eff_conv[i]+=avsum*cf*simpar.ore_density*(1-simpar.porosity)#mol/cm
               ^3_p
    copper.eff_conv*=copper.mmass*simpar.solid_volfrac*simpar.L**3/copper.expected\
       *100
#-------------------------------------------------------------------------------
def getExpectedCopper(copper,simpar,sizeclass,minerals):
    """Calculate the expected amount of copper (in g) from the mineral
    composition of the ore.
    """
    minNames=minerals.keys()
    if 'sulfur' in minNames: minNames.remove('sulfur')
    cpm =[simpar.pstoic[mn][0] for mn in minNames] #moles of copper formed per mole of
       mineral reacted
    copper.expected=0 #total copper expected
    copper.expN={}; #total copper expected from each mineral
    copper.expS={} #total copper expected in each sizeclass
    for sc in sizeclass.keys(): copper.expS[sc]=0
    for cf,name in zip(cpm,minNames):
        copper.expN[name]=0
        for sc in sizeclass.keys():
            mn=minerals[name].grade[sc]*simpar.ore_density*(1-simpar.porosity) #mol/cm
               ^3-p
            copper.expN[name]+=cf*mn*sizeclass[sc].massfrac #mol/cm^3-p
            copper.expS[sc] +=cf*mn*sizeclass[sc].massfrac #mol/cm^3-p
        copper.expected+=copper.expN[name]
    copper.expected*=copper.mmass*simpar.solid_volfrac*simpar.L**3 #g
#-------------------------------------------------------------------------------
def getExpectedIron(simpar,sizeclass,ferric,ferrous):
    pferrous=0;pferric=0
```

```
    for sc in sizeclass.keys():
        pferrous += sizeclass[sc].massfrac*ferrous.mmass*ferrous.pconc #g/L
        pferric += sizeclass[sc].massfrac*ferric.mmass*ferric.pconc #g/L
    piron = (pferrous+pferric)*simpar.porosity*simpar.solid_volfrac*\
        simpar.L**3/1000 #g

    bferrous = ferrous.bulk_conc*ferrous.mmass #g/L
    bferric = ferric.bulk_conc*ferric.mmass #g/L
    biron = (bferrous+bferric)*simpar.bulk_volfrac*simpar.L**3/1000 #g

    fferrous= ferrous.flow_conc*ferrous.mmass
    fferric = ferric.flow_conc*ferric.mmass
    firon = (fferrous+fferric)*simpar.flow_volfrac*simpar.L**3/1000

    simpar.init_total_iron=piron+biron+firon
    #---total iron input by flowing solution
    fferrous_input = ferrous.inletflow_conc*ferrous.mmass #g/L
    fferric_input = ferric.inletflow_conc*ferric.mmass
    simpar.tot_iron_input= (fferrous_input+fferric_input)*simpar.uf*\
        (simpar.L**2)*simpar.time_list[-1]/1000 #g
#-------------------------------------------------------------------------------
def aveAggConc(simpar,sizeclass,species,indx=-1):
    """Calculate the average concentration of a dissolved species (g/cm^3_a) in the
       agglomerate.

    aveAggConc = aveParticleConc + bulkConc + flowConc
    indx is the index in the 'solutions' vector
    """
    pspecies=0;
    for sc in sizeclass.keys():
        intfac=species.psolutions[sc][indx]*sizeclass[sc].r**2
        avspecies=3*simps(intfac,sizeclass[sc].r)/sizeclass[sc].R**3
        pspecies+=avspecies*sizeclass[sc].massfrac #g/cm^3-p x1e3
    pspecies*=simpar.solid_volfrac*(simpar.L**3)/1000 #g
    bspecies=species.bulk_solutions[indx]*simpar.bulk_volfrac*simpar.L**3/1000 #g
    fspecies=species.flow_solutions[indx]*simpar.flow_volfrac*simpar.L**3/1000 #g
    return pspecies,bspecies,fspecies
#**************************************************
from math import exp as mexp
#-----------------------------------------------------------
def getMicrobialSurfaceRate(substrate,datalist,simpar,sizeclass,microbes,oxidizes):
    """Calculate the rate of microbial oxidation of a substrate at the surface of a
       sizeclass.

    substrate = ferrous/sulfur; oxidizes=iron/sulfur
    """
    for mc in microbes.keys():
        ads_pop = datalist[microbes[mc].ads_pop_indx] #cells/g_ore
        if ads_pop < 0 :
            ads_pop=0
            datalist[microbes[mc].ads_pop_indx] = 0
        if microbes[mc].oxidizes==oxidizes:
            for sc in sizeclass.keys():
```

```
                ads_pop_sc = (ads_pop*sizeclass[sc].massfrac) #cells/g_ore_sc
                substrate.surface_rate[sc] = ads_pop_sc*
                    (microbes[mc].ads_growth_rate[sc]/ microbes[mc].myield +\
                        microbes[mc].temperature_function(simpar.temperature)* \
                            microbes[mc].maintenance) #mol/g_ore_sc-min
660             substrate.surface_rate[sc]*=simpar.ore_density*(1-simpar.porosity)#mol/
                    cm^3p_sc-min
#-------------------------------------------------------------
def getMicrobialBulkRate(ferrous,simpar,microbes):
    """Calculate rate of microbial oxidation of ferrous in the bulk phase.
    """
    rate=0
    for mc in microbes.keys():
        if microbes[mc].oxidizes=='iron':
            rate +=microbes[mc].des_pop*(\
                microbes[mc].des_growth_rate/microbes[mc].myield + microbes[mc].\
670                 temperature_function(simpar.temperature)*microbes[mc].maintenance)#
                    mol/Lbulk-min
    ferrous.bulk_rate=rate
#-------------------------------------------------------------
def getAcidGangueRate(datalist,acid,simpar,sizeclass):
    """Calculate the rate of gangue dissolution by acid
    """
    R=8.314 #-> J/mol/K
    fT=acid.ak0*mexp(-acid.Ea/R * (1/simpar.temperature-1/acid.Tref))
    for sc in sizeclass.keys():
        acid_conc = datalist[acid.pconc_indx[sc][0]:acid.pconc_indx[sc][1]]
680     acid.rate[sc]=(simpar.porosity*acid_conc*fT)/1000 #-> mol/cm^3_p
```

Listing B.3: classes.py

```
#!/usr/bin/py
"""Classes module

This module holds all the classes used in the program
"""
#UNITS:
#length=(cm),volume=(L),weight=(g),time=(mins),temperature(K),energy(J)

from Numeric import array,zeros,ones,arange,where,equal
10 from Numeric import exp as nexp
from math import pi,exp as mexp,sqrt as msqrt
from scipy import linspace,isnan
##############################################
class SizeClass:
    def __init__(self,parlist):
        self.N = int(parlist[0]) #grid Number
        self.R = float(parlist[1])/10 #->cm
        self.massfrac = float(parlist[2])
        self.dr = self.R/(self.N-1)
20      self.r = arange(0,self.R+self.dr/2,self.dr)
        #3-D plotting parameters
        self.minFezrange =0
```

```
        self.maxFezrange =0

    def _show(self):
        print '-'*50
        print 'dr =',self.dr*10,'mm'
        print 'Radius =',self.R*10,'mm'
        print 'Mass Fraction =',self.massfrac
        print 'Grid points =',self.N
        print '-'*50

    def __str__(self):
        self._show()
        return ""

    def dMatrix(self,diff,k):
        """Second order central differentiation matrix

        diff=diffusivity,k=mass transfer coefficient
        """
        N=self.N
        m=zeros((N,N),'d')
        a=diff/(2*self.dr)
        for i in xrange(1,N-1):
            m[i,i-1] = 1-1./i
            m[i,i] = -2
            m[i,i+1] = 1+1./i
        m[0,0]= -6; m[0,1]= 6
        m[N-1,N-2]= 2; m[N-1,N-1]= -2-(k/a)*(1 + 1./(N-1))
        m*= diff/self.dr**2
        return m
#-------------------------------------------------------
class Species:
    def __init__(self,parlist,psc):
        for i in xrange(len(parlist)):
            try:
                parlist[i]=float(parlist[i])
            except ValueError:
                pass
        base_pars = 5 # number of parameters common to all species
        if len(parlist)>base_pars:
            #rate of reaction defined in terms of these species,
            #eg ferrous,acid
            self._rspecies=True
        else:
            self._rspecies=False
        self.mmass =parlist[4]
        self.diff =parlist[3]*6*1e5 #->cm^2/min
        self.inletflow_conc =parlist[2]/(self.mmass) #->mol/L
        self.bulk_conc =parlist[1]/(self.mmass) #->mol/L
        self.pconc =parlist[0]/(self.mmass) #->mol/L

        self.flow_conc = self.inletflow_conc #initialize concentration in the flowing
          phase
```

```
        self.diffMatrix={}
        #self.*_solutions are matrices that store values from the simulation.
        self.psolutions={}; self.average_psolutions={}; self.surface_solutions={}
        self.bulk_solutions=[]; self.flow_solutions=[]
        self.pconc_indx={}; self.bulk_indx=0; self.flow_indx=0

        for sc in psc.keys():
            N=psc[sc].N
            self.psolutions[sc] =[]
            self.average_psolutions[sc] =[]
            self.pconc_indx[sc] =[]
            self.surface_solutions[sc] =[]
            self.diffMatrix[sc] =psc[sc].dMatrix(self.diff,self.k)

        if self._rspecies:
            i=base_pars
            self.Ea =parlist[i]; i+=1
            self.ak0 =parlist[i]; i+=1
            self.Tref =parlist[i]+273 #K
            self.rate={}
            for sc in psc.keys():
                N=psc[sc].N
                self.rate[sc]=zeros((N,),'d')

    def getSource(self,name,sc,simpar,minerals,acidrate,microbes=None):
        """Get source term for a species for reactions in the particle
        using reaction stoichiometry and formula rates
        """
        if name == 'copper':
            ind=0
        elif name == 'ferric':
            ind=1
        elif name=='ferrous':
            ind=2
        elif name=='acid':
            ind=3
        minNames=minerals.keys();
        if 'sulfur' in minNames: minNames.remove('sulfur')
        if name == 'oxygen':
            N = len(acidrate[sc])
            source=zeros((N,),'d') #return initial source which is a vector of zeros
        else:
            source=0
            for m in minNames:
                source=source+simpar.pstoic[m][ind]*minerals[m].rate[sc]
            if name == 'acid':
                source=source+simpar.pstoic['gangue'][ind]*acidrate[sc]
    return source

    def getSurfaceSource(self,name,simpar,ferrous_srate,sulfur_srate):
        """Get source term for a species for reactions at the particle surface
        using reaction stoichiometry and microbial oxidation rates
        """
```

```
            #index 0 =ferrous; index 1 = sulfur. See simpar.bstoic
            source =simpar.bstoic[name][0]*ferrous_srate+\
                simpar.bstoic[name][1]*sulfur_srate
            return source

        def _show(self):
            print '-'*50
            print 'Molar mass =',self.mmass,' (g/mol)'
            print 'Diffusivity = %g (cm^2/min)'% self.diff
            print 'Bulk concentration = %g (mol/L)'% (self.bulk_conc)
            print 'Flowing liq. conc = %g (mol/L)'% (self.flow_conc)
            if self._rspecies:
                print 'Activation Energy = %g (J/mol)'% self.Ea
                print 'Arrhenius Constant =',self.ak0,' (1/min)'
                print 'Ref. Temperature =',self.Tref,' (K)'
            print '-'*50

        def __str__(self):
            self._show()
            return ""
    #-------------------------------------------------------------------
    class Minerals:
        def __init__(self,parlist,psc):
            for i in xrange(2,len(parlist)): #do not include grades
                try:
                    parlist[i]=float(parlist[i])
                except ValueError:
                    pass
            self.type = parlist[0]
            self.mmass = parlist[2]
            self.phi = parlist[3]
            self.Ea = parlist[4]
            self.ak0 = parlist[5]
            self.Tref = parlist[6]+273 #->K
            self.pstoic = parlist[7]
            if self.type == 'sulfide':
                self.m = parlist[8]
                self.ka = parlist[9] #mol/L
                self.kb = parlist[10] #mol/L
                self.Sstoic = parlist[11] #->sulfur stoichiometry in the mineral leaching
                    reaction

            #sort out the grades
            grade=parlist[1].split(',')
            grade_list=[float(i) for i in grade]

            #check that number of grades == number of sizeclasses.
            #Does not matter if it is more as the extra values are ignored.
            if len(grade_list) < len(psc):
                from engine import errorHandler
                message="Grade list is less than number of particle size classes"
                errorHandler(message)
```

```
        #rate,conversion,solution matrix,grade
        self.grade={};self.rate={};
        self.solutions={};self.average_solutions={}
        self.conv_indx={}

        i=0
        keys=psc.keys()
        keys.sort()
        for sc in keys:
            self.grade[sc]=grade_list[i]; i+=1
            self.rate[sc]=zeros((psc[sc].N,),'d')
            self.solutions[sc]=[]
            self.average_solutions[sc]=[]
            self.conv_indx[sc] = []
        #NOTE that correct assignment of grades is dependent on the
        #psckeys being sorted in such a way that it matches the order
        #of arrangement of the grades. Hence sizeclass names should
        #start the same and end with integers appended in increasing order.

    def updateRxnRate(self,datalist,simpar,sizeclass,fe3,fe2,acid):
        """Calculate the rate of mineral conversion.
        """
        R=8.314 #J/mol/K
        fT=self.ak0 * mexp(-self.Ea/R * (1/simpar.temperature - 1/self.Tref))
        for sc in sizeclass.keys():
            fx = (1-datalist[self.conv_indx[sc][0]:self.conv_indx[sc][1]])**self.phi
            #sometimes, conversion (X) goes >1 due to numerics. Force X=1 if X>1
            if isnan(fx):
                temp=isnan(fx)
                eq0_ind=equal(temp,1)
                fx=where(eq0_ind,0.0,fx)
            if self.type == 'sulfide':
                fe3conc = datalist[fe3.pconc_indx[sc][0]:fe3.pconc_indx[sc][1]]
                fe2conc = datalist[fe2.pconc_indx[sc][0]:fe2.pconc_indx[sc][1]]
                fc= fe3conc/( (self.ka+fe3conc)**(1-self.m) * \
                    (self.kb+fe2conc)**self.m )
            elif self.type == 'oxide':
                fc= datalist[acid.pconc_indx[sc][0]:acid.pconc_indx[sc][1]]
            self.rate[sc]=(1-simpar.porosity)*simpar.ore_density*\
                self.grade[sc]*fT*fx*fc #mol/cm^3_p

    def _show(self):
        print '-' * 50
        print 'Molar mass =',self.mmass,'(g/mol)'
        print 'Topological Exponent(phi) =',self.phi
        print 'Activation Energy = %g (J/mol)'% self.Ea
        print 'Arrhenius Constant =', self.ak0, ' (1/min)'
        print 'Ref. Temperature =',self.Tref,'(K)'
        if self.type == 'sulfide':
            print 'Ferric reduction exponent(m) =', self.m
            print 'Ferric mass transfer parameter(ka) = %g (mol/L-water)' %self.ka
            print 'Ferric reduction parameter(kb) = %g (mol/L-water)' %self.kb
        print 'Grades in sizeclass'
```

```
        sckeys=self.grade.keys()
        sckeys.sort()
        for sc in sckeys:
            print "%s \t= %f (mol/g-ore)" %(sc,self.grade[sc])
        print '-' * 50

    def __str__(self):
        self._show()
        return ""
#--------------------------------------------------------------------
class Simpar:
    def __init__(self,parlist):
        for i in xrange(len(parlist)):
            try:
                parlist[i]=float(parlist[i])
            except ValueError:
                pass
        i=0
        #physical parameters
        self.ore_density = parlist[i]; i+=1
        self.porosity = parlist[i]; i+=1
        self.bulk_volfrac = parlist[i]; i+=1
        self.flow_volfrac = parlist[i]; i+=1
        self.temperature = parlist[i]+273; i+=1 #->K
        self.L = parlist[i]; i+=1
        self.uf = parlist[i] ; i+=1 #superficial velocoty (cm/min)
        self.ksb = parlist[i]; i+=1 #mass transfer coefficient for part/stag interface
           (cm/min)
        self.kbf = parlist[i] ; i+=1 #mass transfer rate for stag/flow interface (1/min
           )
        #simulation parameters
        self.tstop = parlist[i]*60; i+=1 #->mins
        self.timesteps = parlist[i]; i+=1
        self.plot_gap = parlist[i]; i+=1
        self.report_style = parlist[i]; i+=1
        self.outdirname = parlist[i]; i+=1
            #boolean parameters
        self.plot2d = parlist[i]; i+=1
        self.plot3d = parlist[i]; i+=1
        self.constant_in_bulk={}
        self.constant_in_bulk['ferric'] =parlist[i]; i+=1
        self.constant_in_bulk['ferrous']=parlist[i]; i+=1
        self.constant_in_bulk['acid'] =parlist[i]; i+=1
        self.constant_in_bulk['oxygen'] =parlist[i]; i+=1
        self.constant_in_bulk['copper'] ='no'
        self.microbes = parlist[i]; i+=1 #there are microbes in the system
        self.solid_volfrac = 1-(self.flow_volfrac+self.bulk_volfrac)

        self.time_list=linspace(0,self.tstop,self.timesteps)
        self.pstoic={} #stoichiometry matrix for particle reactions
        self.bstoic={} #stoichiometry vector for bulk reaction
        self.outfiles={} #files that store simulation results
```

```
    def getpstoic(self,minerals):
        """Stoichiometry matrix for reactions in the ore particles
        """
        stoic={}
        if 'pyrite' in minerals.keys():
            beta=minerals['pyrite'].Sstoic
        else:
            beta=0
        minNames=minerals.keys()
        if 'sulfur' in minNames: minNames.remove('sulfur')
         # array columns: Cu Fe3+ Fe2+ Acid
        stoic['gangue'] = array([0.0, 0.0, 0.0,-1.0],'d')
        for mn in minNames:
            tt=minerals[mn].pstoic.split(',')
            stoic[mn]=array([eval(i) for i in tt],'d')
        return stoic

    def getbstoic(self):
        """Stoichiometry matrix for reactions in the bulk solution
        """
        stoic={}
        # Fe2+ S
        stoic['copper'] = [0. , 0.]
        stoic['ferric'] = [1.0, 0.]
        stoic['ferrous']= [-1.0, 0.]
        stoic['acid'] = [-0.5, 1.]
        stoic['oxygen'] = [-0.25, -1.5]
        return stoic

    def getLogs(self):
        """Format log from simulation run.
        """
        import string
        from scipy.integrate import odeint
        doc = odeint.__doc__
        #select the required section of the doc string
        lind = string.find(doc,'infodict --')
        uind = string.find(doc,'Additional')
        log='\n'+ '.'*60 + '\n'
        log+="An additional dictionary of outputs is returned by the integration function '
           scipy.integrate.odeint'. The entries in this dictionary are explained in the
           excerpt below (from the documentation of odeint):" + '\n\n'
        log+=doc[lind:uind]
        log+="Additional outputs from simulation" + '\n'
        keys=self.infodict.keys()
        keys.sort()
        for k in keys:
            log += k + ': ' + str(self.infodict[k]) + '\n'
        self.log = log

    def _show(self):
        print '-' * 50
        print "Simulation time = %g (hrs)" % (self.tstop/60)
```

```
        print "Timestep = ",self.dt," (mins)"
        print "Plot interval = %g (mins)" % self.plot_interval
        print "Ore density = %g (g/cm^3)" % self.ore_density
        print "Agglomerate temperature = %g (K)" % self.temperature
        print "Particle porosity = %g " % self.porosity
        print "Solid to liquid ratio (volume) = %g" % self.vsol_liq
        print '-' * 50

    def __str__(self):
        self._show()
        return ""
#------------------------------------------------------------------------
class Microbes:
    from math import sqrt as msqrt
    def __init__(self,parlist,sizeclass,simpar):
        for i in xrange(len(parlist)):
            try:
                parlist[i]=float(parlist[i])
            except ValueError:
                pass
        i=0
        self.oxidizes =parlist[i]; i+=1
        self.bulk_pop =parlist[i]; i+=1 #->cells/L
        self.inlet_flow_pop =parlist[i]; i+=1 #->cells/L
        #***physical parameters
        self.mmass =parlist[i]; i+=1
        self.diff =parlist[i]*6*1e5; i+=1 #->cm^2/min
        self.myield =parlist[i]; i+=1
        self.maintenance =parlist[i]; i+=1
        self.max_growth_rate=parlist[i]; i+=1
        self.death_rate =parlist[i]; i+=1
        #***adsorption parameters(Langmuir)
        self.ads_max =parlist[i]; i+=1
        self.adsEq_const =parlist[i]; i+=1
        #temperature parameters(Ratkowsky)
        self.Tmin =parlist[i]+273; i+=1 #-> K
        self.Tmax =parlist[i]+273; i+=1
        self.Topt =parlist[i]+273; i+=1
        self.b =parlist[i]; i+=1
        self.c =parlist[i]; i+=1
        #Monod parameters
        self.inhib_fac =parlist[i]; i+=1
        self.oxygen_monod =parlist[i]; i+=1
        if self.oxidizes == 'iron':
            self.ferrous_monod =parlist[i]; i+=1
            self.acidlim =parlist[i]; i+=1
        elif self.oxidizes=='sulfur':
            self.sulfur_monod =parlist[i]; i+=1

        self.ads_growth_rate={}
        self.des_growth_rate=0

        self.ads_pop_indx=0; self.des_pop_indx=0; self.flow_indx=0
```

```
        #store microbial growth rate at certain intervals
        self.agr_solutions={} #adsorbed growth rate
        self.dgr_solutions=[] #desorbed growth rate

        for sc in sizeclass.keys():
            self.ads_growth_rate[sc]=0
            self.agr_solutions[sc]=[]
        self.bulk_pop_solutions=[]
        self.ads_solutions =[]
        self.des_solutions =[]
        self.flow_solutions=[]

        self.flow_pop=self.inlet_flow_pop #arbitrarily initialize cells in the flowing
           phase

        unitConv=simpar.ore_density*(1-simpar.porosity)*simpar.solid_volfrac*\
            1000/simpar.bulk_volfrac #[cells/g_ore->cells/Lb]
        temp1 =1+self.adsEq_const*(self.bulk_pop+unitConv*self.ads_max)
        temp2 =msqrt(temp1**2 - 4*unitConv*self.ads_max*self.bulk_pop*\
            self.adsEq_const**2)
        self.ads_pop= (temp1-temp2)/(2*unitConv*self.adsEq_const) #cells/g_ore
        self.des_pop=self.bulk_pop-(unitConv*self.ads_pop)
        #calculate optimum temperature parameter
        self.kopt = self.b*(self.Topt-self.Tmin) * (1-mexp(self.c * \
            (self.Topt-self.Tmax) ))

    def temperature_function(self,temperature):
        if (temperature > self.Tmax) or (temperature < self.Tmin):
            return 0
        else:
            ktemp= self.b*(temperature-self.Tmin) * (1-mexp(self.c * \
                (temperature-self.Tmax) ))
            return ktemp/self.kopt

    def applyLangmuir(self,datalist,simpar):
        """Calculate adsorbed/desorbed partitioning from Langmuir Isotherm.
        """
        unitConv=simpar.ore_density*(1-simpar.porosity)*simpar.solid_volfrac*\
            1000/simpar.bulk_volfrac #[cells/g_ore->cells/Lb]
        temp1 =1+self.adsEq_const*(self.bulk_pop+unitConv*self.ads_max)
        temp2 =msqrt(temp1**2 - 4*unitConv*self.ads_max*self.bulk_pop*\
            self.adsEq_const**2)
        self.ads_pop= (temp1-temp2)/(2*unitConv*self.adsEq_const) #cells/g_ore
        if temp1<temp2:
            sys.exit()
        self.des_pop=self.bulk_pop-(unitConv*self.ads_pop) #cells/Lb
        datalist[self.ads_pop_indx] = self.ads_pop
        datalist[self.des_pop_indx] = self.des_pop

    def updateGrowthRate(self,datalist,simpar,sizeclass,species,sulfur):
        if self.oxidizes == 'iron':
            #growth rate of desorbed microbes
```

```
            #--get bulk indicies
            acidbindx = species['acid'].bulk_indx
            fe2bindx=species['ferrous'].bulk_indx
            oxbindx=species['oxygen'].bulk_indx
            #--get growth limitation term
            acid_term = 1-mexp(-datalist[acidbindx]/self.acidlim)
            ferrous_term = datalist[fe2bindx]/(self.ferrous_monod +\
                datalist[fe2bindx])
            oxygen_term = datalist[oxbindx]/(self.oxygen_monod +\
                datalist[oxbindx])
            growth_lim = acid_term * ferrous_term * oxygen_term
            #--get growth inhibition term
            growth_inhib = self.inhib_fac/(self.bulk_pop+self.inhib_fac)
            #--growth rate
            self.des_growth_rate = self.max_growth_rate* growth_lim * \
                growth_inhib *self.temperature_function(simpar.temperature)
            #growth of adsorbed microbes
            for sc in sizeclass.keys():
                #--get surface indicies
                acidsindx = species['acid'].pconc_indx[sc][1]-1
                fe2sindx = species['ferrous'].pconc_indx[sc][1]-1
                oxsindx = species['oxygen'].pconc_indx[sc][1]-1
                #--get growth limitation terms
                acid_term = 1-mexp(-datalist[acidsindx]/self.acidlim)
                ferrous_term = datalist[fe2sindx]/(self.ferrous_monod + \
                    datalist[fe2sindx])
                oxygen_term = datalist[oxsindx]/(self.oxygen_monod + \
                    datalist[oxsindx])
                growth_lim = acid_term * ferrous_term * oxygen_term
                #--growth rate (inhibition term same as for desorbed microbes)
                self.ads_growth_rate[sc] =self.max_growth_rate * growth_lim * \
                    growth_inhib *self.temperature_function(simpar.temperature)
                #--microbes in particles
        elif self.oxidizes == 'sulfur':
            #growth rate of adsorbed microbes
            for sc in sizeclass.keys():
                #--get surface indicies
                sulfursindx = sulfur.grade_indx[sc][1]-1
                oxsindx =species['oxygen'].pconc_indx[sc][1]-1
                #--get growth limitation terms
                sulfur_term = datalist[sulfursindx]/(self.sulfur_monod + \
                    datalist[sulfursindx])
                oxygen_term = datalist[oxsindx]/(self.oxygen_monod + \
                    datalist[oxsindx])
                growth_lim = sulfur_term * oxygen_term
                #--get growth inhibition term
                growth_inhib = self.inhib_fac/(self.bulk_pop+self.inhib_fac)
                #growth rate
                self.ads_growth_rate[sc] =self.max_growth_rate* growth_lim*\
                    growth_inhib *self.temperature_function(simpar.temperature)
                #--microbes in particles

    def _show(self):
```

```
        print '-' * 50
        print "Type: %s oxidizing %s" % (self.oxidizes,self.type)
        print "Population = %g (cells/cm^3 agg)" % self.agg_pop
        print "***Physical Parameters***"
        print "Molar mass = %g (g/mol)" % self.mmass
        print "Difusivity = %g (cm^2/min)" % self.diff
        print "Yield = %g (cells/mol Fe2+)" % self.myield
        print "Maintenance rate = %g (mol Fe2+/cell-min)" % self.maintenance
        print "Maximum growth rate = %g (1/min)" % self.max_growth_rate
        print "Death rate = %g (1/min)" % self.death_rate
        print "***Adsorption Parameters(Langmuir)***"
        print "Max. adsorbable microbes = %g (cells/g-ore)" % self.ads_max
        print "Adsorption equilibrium constant= %g (L/cell)" % self.adsEq_const
        print "***Temperature Parameters(Ratkowsky)***"
        print "Tmin = %g (K)" % self.Tmin
        print "Tmax = %g (K)" % self.Tmax
        print "Topt = %g (K)" % self.Topt
        print "Fitting parameter b = %g (K)" % self.b
        print "Fitting parameter b = %g (K)" % self.c
        print "***Monod parameters***"
        print "Inhibition constant = %g (mol/L)" % self.inhib_fac
        print "Oxygen Monod constant = %g (mol/L)" % self.oxygen_monod
        if self.oxidizes == 'iron':
            print "Ferrous Monod constant = %g (mol/L)" % self.ferrous_monod
            print "Acid limiting constant = %g (mol/L)" %self.acidlim
        elif self.oxidizes == 'sulfur':
            print "Sulfur Monod constant = %g (mol/L)" % self.sulfur_monod
        print '-' * 50

    def __str__(self):
        self._show()
        return ""
#-----------------------------------------------
class SulfurMineral(Minerals):
    def __init__(self,psc):
        self.grade={};self.solutions={};self.average_solutions={}
        self.type=None
        self.mmass=32
        self.surface_rate={}; self.prate={};self.grade_indx={}

        keys=psc.keys()
        keys.sort()
        for sc in keys:
            self.solutions[sc]=[]
            self.average_solutions[sc]=[]
            self.surface_rate[sc]=0
            self.grade_indx[sc]=[]

    def _show(self):
        print '-'*50
        print "Molar mass = ", self.mmass
        print '-'*50
```

```
    def __str__(self):
        self._show()
        return ""
```

Listing B.4: IO.py

```
#!/usr/bin/python
"""Input/Output functions

This module contains input/output functions used by the program
"""

from string import split,strip,replace,rstrip
import os
import Gnuplot
from Numeric import ceil
from engine import errorHandler,tryCommand
from classes import *

def checkBoolean(parameter,name=''):
    """Verify the value of a boolean parameter.

    A boolean parameter in the input files takes the value 'yes' or 'no'.
    If current value is neither, set it to 'yes'
    """
    if not (parameter in ['yes','no']):
        parameter=True
        print "** %s parameter set to 'yes'**\n" % name
    elif parameter == 'yes':
        parameter=True
    elif parameter == 'no':
        parameter=False
    return parameter
#--------------------------------------------------------------------
def readParFile(filename):
    """Read an input file, and return a dictionary of parameter values.

    usage: parDict=readParFile(filename)
    """
    try:
        parfile=open(filename,'r')
    except IOError:
        message="No such file %s" % filename
        errorHandler(message)

    parameter_dict={}
    go_on=True
    while go_on:
        line=parfile.readline()
        if line == "":
            go_on=False
        elif line[0] == '#':
            #ignore comment lines
```

```
            pass
        elif 'begin' in line:
            temp=[]
            name=strip(line.split()[1])
            parameter_dict[name]=0
            #read the remaining lines till "end"
            line=parfile.readline()
            while not "end" in line:
                #ignore comment lines
                if line[0]=="#":
                    pass
                else
                    temp.append(strip(line.split('=')[1]))
                line=parfile.readline()
            #store values
            parameter_dict[name]=temp[:]
    parfile.close()
    return parameter_dict
#-----------------------------------------------------------
def initializeFromFiles(simpar_file,psc_file,minerals_file,\
        species_file,microbes_file,dir):
    """Read parameters from input files, create the class instances
     from these values,and return each instance as an entry in the
     associated dictionary.

    returns simpar,sizeclassDict,mineralsDict,speciesDict.
    simpar is just a class instance. The other dictionaries have keys
    defined by the user in the input file using the 'begin'
    keyword.

    All input files must be in the same directory, set by 'dir'.
    Default is "cwd/input-files" where cwd=current working directory.
    """
    #---initalize simulation parameters---#
    simpar_file=os.path.join(dir,simpar_file)
    temp_dict=readParFile(simpar_file)
    key = temp_dict.keys()
    simpar=Simpar(temp_dict[key[0]])
    #check that volume fractions add up to 1
    if (simpar.bulk_volfrac +simpar.flow_volfrac) >= 1:
        message="(stagnant phase)+(flowing phase) volume fraction is >= 1"
        errorHandler(message)

    #check that particle porosity > 0
    if simpar.porosity <= 0:
        message="Particle porosity must be > 0"
        errorHandler(message)
    #check boolean parameters
    simpar.plot2d =checkBoolean(simpar.plot2d,name='plot2d')
    simpar.plot3d =checkBoolean(simpar.plot3d,name='plot3d')
    simpar.microbes =checkBoolean(simpar.microbes,name='microbes')
    for sn in simpar.constant_in_bulk.keys():
        simpar.constant_in_bulk[sn]=checkBoolean(simpar.constant_in_bulk[sn],name=sn)
```

```
    #---initialize particle size classes---#
    psc_file=os.path.join(dir,psc_file)
    temp_dict=readParFile(psc_file)
    sizeclass={}
    for sc in temp_dict.keys():
        sizeclass[sc]=SizeClass(temp_dict[sc])

    #check that total mass fraction=1
    tot_mass_frac=0
    for sc in sizeclass.keys():
        tot_mass_frac+=sizeclass[sc].massfrac
    if tot_mass_frac != 1:
        message="Total mass fraction must be = 1"
        errorHandler(message)

    #---initialize minerals---#
    minerals_file=os.path.join(dir,minerals_file)
    temp_dict=readParFile(minerals_file)
    minerals={};
    simpar.sulfur_present=False;
    for mn in temp_dict.keys():
        minerals[mn]=Minerals(temp_dict[mn],sizeclass)
        if minerals[mn].type == 'sulfide': simpar.sulfur_present=True

    #Initialize Sulfur mineral
    if simpar.sulfur_present:
        minerals['sulfur']=SulfurMineral(sizeclass)

    #***set up stoichiometry matrix
    simpar.pstoic=simpar.getpstoic(minerals) #->particle reactions stoichiometry
    simpar.bstoic=simpar.getbstoic() #->bulk reactions stoichiometry

    #---initialize chemical species---#
    Species.k = simpar.ksb
    species_file=os.path.join(dir,species_file)
    temp_dict=readParFile(species_file)
    species={}
    for sn in temp_dict.keys():
            species[sn]=Species(temp_dict[sn],sizeclass)

    #---initialize microbial species---#
    microbes={}
    if simpar.microbes:
        microbes_file=os.path.join(dir,microbes_file)
        temp_dict=readParFile(microbes_file)
        if temp_dict.keys(): #Microbes file has an entry
            for mc in temp_dict.keys():
                if not temp_dict[mc][0] in ["iron","sulfur"]:
                    message="Microbe must be 'iron' or 'sulfur' oxidizer\n
                    Check 'oxidizes' parameter in %s" % microbes_file
                    errorHandler(message)
                if not temp_dict[mc][1] in ["mesophile","thermophile"]:
```

```
                    message="Microbe type must be mesophile or thermophile\n\
                    Check 'type' parameter in %s" % microbes_file
                    errorHandler(message)
                microbes[mc]=Microbes(temp_dict[mc],sizeclass,simpar)
        else: #Microbes file has no entry
            simpar.microbes = False
            print "Microbes file has no entry. Running abiotic simulation."
    return simpar,sizeclass,minerals,species,microbes
#-----------------------------------------------------------------
def createDataFiles(simpar,min_names,species_names,microbe_names,\
        sizeclass_names,outdir):
    """Create files that hold simulation results and simulation logs.
    """
    #3D files
    simpar.outfiles['files3d']={}
    header2d=""
    sizeclass_names.sort()
    for sc in sizeclass_names:
        header2d+="\t"+sc
        simpar.outfiles['files3d'][sc]={}
        for mn in min_names:
            filename=mn+'3D-'+sc+'.txt'
            filename=os.path.join(outdir,filename)
            simpar.outfiles['files3d'][sc][mn]=open(filename,'w')
            #write header
            if mn == 'sulfur':
                header="#Radius(mm) \ttime(hrs) \tGrade(g/g-ore)\n"
            else:
                header="#Radius(mm) \ttime(hrs) \tConversion(%)\n"
            simpar.outfiles['files3d'][sc][mn].write(header)
        for sn in species_names:
            filename=sn+'3D-'+sc+'.txt'
            filename=os.path.join(outdir,filename)
            simpar.outfiles['files3d'][sc][sn]=open(filename,'w')
            #write header
            header="#Radius(mm) \ttime(hrs) \tConc(g/cm^3 part)Xe-3\n"
            simpar.outfiles['files3d'][sc][sn].write(header)
    header2d+="\n"
    #2D files
    simpar.outfiles['files2d']={}
    simpar.outfiles['files2d']['bulkConc']={} #bulk concentration
    simpar.outfiles['files2d']['flowConc']={} #concentration in flowing solution
    simpar.outfiles['files2d']['avePsolns']={} #average particle concentration
    simpar.outfiles['files2d']['surfaceConc']={}

    filename=os.path.join(outdir,'SpeciesBulkConc.txt')
    simpar.outfiles['files2d']['bulkConc']['species']=open(filename,'w')
    filename=os.path.join(outdir,'MicrobialBulkConc.txt')
    simpar.outfiles['files2d']['bulkConc']['microbes']=open(filename,'w')
    filename=os.path.join(outdir,'SpeciesFlowConc.txt')
    simpar.outfiles['files2d']['flowConc']['species']=open(filename,'w')
    filename=os.path.join(outdir,'MicrobialFlowConc.txt')
    simpar.outfiles['files2d']['flowConc']['microbes']=open(filename,'w')
```

```
header='#'+"%16s" % ("time(hrs)"); header2d=header+header2d
for sn in species_names:
    header+="%15s" % (sn+"(g/L-bulk)")
    filename=sn+'2D.txt'
    filename=os.path.join(outdir,filename)
    simpar.outfiles['files2d']['avePsolns'][sn]=open(filename,'w')
    simpar.outfiles['files2d']['avePsolns'][sn].write(header2d)
    #**
    filename=sn+'SurfConc.txt'
    filename=os.path.join(outdir,filename)
    simpar.outfiles['files2d']['surfaceConc'][sn]=open(filename,'w')
    simpar.outfiles['files2d']['surfaceConc'][sn].write(header2d)
simpar.outfiles['files2d']['bulkConc']['species'].write(\
    header+"-- AvCopperConv(%)"+"\n")
simpar.outfiles['files2d']['flowConc']['species'].write(header+"\n")

header1='#'+"%16s" % ("time(hrs)"); header2='#'+"%16s" % ("time(hrs)")
for mc in microbe_names:
    header1+="%15s" % ("Tot"+mc+"(c/cm3a)"); header2+="%15s" % \
        ("Tot"+mc+"(c/cm3a)")
    header1+="%15s"% ("Des"+mc+"(c/Lb)")
    header1+="%15s" % ("Ads"+mc+"(c/Lb)")
header1 +="\n"; header2 +="\n"
simpar.outfiles['files2d']['bulkConc']['microbes'].write(header1)
simpar.outfiles['files2d']['flowConc']['microbes'].write(header2)

for mn in min_names:
    filename=mn+'2D.txt'
    filename=os.path.join(outdir,filename)
    simpar.outfiles['files2d']['avePsolns'][mn]=open(filename,'w')
    simpar.outfiles['files2d']['avePsolns'][mn].write(header2d)
simpar.outfiles['misc']={}
#file for simulation logs
filename=os.path.join(outdir,'simlogs.log')
simpar.outfiles['misc']['simlog']=open(filename,'w')
#other files
filename=os.path.join(outdir,'MicrobialGrowthRate.txt')
simpar.outfiles['misc']['mgrate']=open(filename,'w')
header_mrate = replace(header2d,"(g/L-bulk)","1/min" )
header_mrate = rstrip(header_mrate)
header_mrate+= "%15s" % "bulk\n#"
for mc in microbe_names:
    header_mrate += "%40s" % mc
header_mrate+="\n"

simpar.outfiles['misc']['mgrate'].write(header_mrate)
filename=os.path.join(outdir,'ferrousOxRate.txt')
simpar.outfiles['misc']['fe2rate']=open(filename,'w')
header_frate = replace(header2d,"(g/L-bulk)","(g/Lb-min)")
header_frate = rstrip(header_frate)
header_frate+= "%15s" % "bulk\n"
simpar.outfiles['misc']['fe2rate'].write(header_frate)
```

```
260 #-------------------------------------------------------------
    def closeDataFiles(datafiles):
        """Close files that contain simulation results
        """
        for sc in datafiles['files3d'].keys():
            for name in datafiles['files3d'][sc].keys():
                datafiles['files3d'][sc][name].close()
        for name in datafiles['files2d']['bulkConc'].keys():
            datafiles['files2d']['bulkConc'][name].close()
        for name in datafiles['files2d']['flowConc'].keys():
270         datafiles['files2d']['flowConc'][name].close()
        for name in datafiles['files2d']['avePsolns'].keys():
            datafiles['files2d']['avePsolns'][name].close()
        for name in datafiles['files2d']['surfaceConc'].keys():
            datafiles['files2d']['surfaceConc'][name].close()
        datafiles['misc']['mgrate'].close()
        datafiles['misc']['fe2rate'].close()
    #-------------------------------------------------------------
    def writeData3d(file,radius,time,data):
        for tt in xrange(len(time)):
280         for ii in xrange(len(radius)):
                dataset="%.5f \t %.5f \t %.5f \n" % \
                    (radius[ii]*10,time[tt]/60,data[tt][ii])
                file.write(dataset)
            file.write('\n')
    #-------------------------------------------------------------
    def writeMisc(simpar,ferrous,microbes):
        sckeys=ferrous.sr_solutions.keys()
        sckeys.sort()
        time = simpar.time_list
290     for tt in range(len(time)):
            fdataset = "%15.4f" % (time[tt]/60)
            mdataset = "%15.4f" % (time[tt]/60)
            for sc in sckeys:
                fdataset += "%15.5g" % (ferrous.sr_solutions[sc][tt])
            fdataset += "%15.5g\n" % (ferrous.br_solutions[tt])
            simpar.outfiles['misc']['fe2rate'].write(fdataset)
            for mn in microbes.keys():
                for sc in sckeys:
                    mdataset += "%15.5g" % (microbes[mn].agr_solutions[sc][tt])
300             mdataset += "%15.5g" % (microbes[mn].dgr_solutions[tt])
            mdataset += "\n"
            simpar.outfiles['misc']['mgrate'].write(mdataset)
    #-------------------------------------------------------------
    def writeData2d(files2d,time,species,microbes,minerals):
        sizeclass_keys=species['copper'].psolutions.keys()
        sizeclass_keys.sort()
        for tt in xrange(len(time)):
            bulk_dataset="%15.4f" % (time[tt]/60)
            flow_dataset="%15.4f" % (time[tt]/60)
310         for sn in species.keys():
                bulk_dataset+="%15.5f" % species[sn].bulk_solutions[tt]
                flow_dataset+="%15.5f" % species[sn].flow_solutions[tt]
```

```
            ave_dataset= "%15.5f" % (time[tt]/60)
            surf_dataset="%15.5f" % (time[tt]/60)
            for sc in sizeclass_keys:
                ave_dataset+=" %15.5f" % species[sn].average_psolutions[sc][tt]
                surf_dataset+="%15.5f" % species[sn].surface_solutions[sc][tt]
            ave_dataset+= "\n"; surf_dataset+="\n"
            files2d['avePsolns'][sn].write(ave_dataset)
            files2d['surfaceConc'][sn].write(surf_dataset)
        bulk_dataset+="%15.5f" % species['copper'].eff_conv[tt]
        bulk_dataset+="\n";flow_dataset+="\n"
        files2d['bulkConc']['species'].write(bulk_dataset)
        files2d['flowConc']['species'].write(flow_dataset)

        #microbial data
        bulk_dataset="%10.4f " % (time[tt]/60)
        flow_dataset="%10.4f " % (time[tt]/60)
        for sn in microbes.keys():
            bulk_dataset+=" %15.7f %15.7f %15.7f" % \
                (microbes[sn].bulk_pop_solutions[tt],microbes[sn].des_solutions[tt],\
                    microbes[sn].ads_solutions[tt])
            flow_dataset+=" %15.7f " % microbes[sn].flow_solutions[tt]
        bulk_dataset+="\n";flow_dataset+="\n"
        files2d['bulkConc']['microbes'].write(bulk_dataset)
        files2d['flowConc']['microbes'].write(flow_dataset)

        for mn in minerals.keys():
            ave_dataset= "%15.4f " % (time[tt]/60)
            for sc in sizeclass_keys:
                ave_dataset +=" %15.5f" % minerals[mn].average_solutions[sc][tt]
            ave_dataset+= "\n"
            files2d['avePsolns'][mn].write(ave_dataset)
#-------------------------------------------------------------
def plot3d(plot_gap,datafiles,speciesNames,microbesNames,sizeclass,fileType,outdir):
    if not(fileType in ['eps','png']) :
        message= "fileType must be eps or png"
        errorHandler(message)
    g=Gnuplot.Gnuplot(debug=0)
    g('set style data lines')
    g('unset key')
    g('set view 60,36')
    g('set grid ztics')
    g('set border 1023-128')
    g.ylabel('time(hrs)')
    flist=[]
    if fileType=='eps':
        g.xlabel('radius(mm)&{junk}')
        for sc in datafiles.keys():
            smin=min(sizeclass[sc].r*10); smax=max(sizeclass[sc].r*10) #->mm
            g('set xrange [%g:%g]'%(smin,smax))
            for key in datafiles[sc].keys():
                if key in speciesNames:
                    g('set zlabel "Conc. x 10^{-3} (g/cm^3 particle) "')
                elif key in microbesNames:
```

```
                g('set zlabel "Pop. x 10^{ 3} (cells/cm^3 particle)"')
            elif 'sulfur' in key:
                g('set zlabel "Grade (g/g ore)"')
            else:
                g('set zlabel "Frac. reacted (%)"')
            #get output filename
            fname=datafiles[sc][key].name
            title=os.path.basename(fname)
            outfile=os.path.join(outdir,title)
            outfile=outfile.replace('txt','eps')
            (title, extension) = os.path.splitext(title)
            flist.append(outfile)
            if 'ferric' in key or 'ferrous' in key:
                if sizeclass[sc].maxFezrange<3:
                    a='%e' % sizeclass[sc].maxFezrange;a=a.split('e')
                    a0=ceil(float(a[0]));a1=float(a[1])
                    newmax=(a0*10**a1)
                    a=newmax/a0
                    g('set ztics %s' % str(a))
                    g('set zrange [%g:%g]' % (sizeclass[sc].minFezrange,newmax))
                else:
                    g('set zrange [%g:%g]' % (sizeclass[sc].minFezrange,\
                        sizeclass[sc].maxFezrange))
            g('set term postscript eps enhanced 15')
            g('set title "%s"' % title)
            g("set output '%s'" % outfile)
            g("splot '%s' every :%s" % (fname,plot_gap))
            g('set output')
            g('set autoscale z')
            g('set ztics autofreq')
    else: #fileType==png
        g.xlabel('radius(mm)')
        for sc in datafiles.keys():
            smin=min(sizeclass[sc].r*10); smax=max(sizeclass[sc].r*10) #->mm
            g('set xrange [%g:%g]'%(smin,smax))
            for key in datafiles[sc].keys():
                if key in speciesNames:
                    g('set zlabel "Conc x 1e-3 (g/cm^3 p)"')
                elif key in microbesNames:
                    g('set zlabel "Pop x 1e-3 (cells/cm^3 p)"')
                elif 'sulfur' in key:
                    g('set zlabel "Grade (g/g ore)"')
                else:
                    g('set zlabel "Frac. reacted (%)"')
                #get output filename
                fname=datafiles[sc][key].name
                title=os.path.basename(fname);
                outfile=os.path.join(outdir,title)
                outfile=outfile.replace('txt','png')
                (title, extension) = os.path.splitext(title)
                flist.append(outfile)
                if 'ferric' in key or 'ferrous' in key:
                    if sizeclass[sc].maxFezrange<3:
```

```
                    a='%e' % sizeclass[sc].maxFezrange;a=a.split('e')
                    a0=ceil(float(a[0]));a1=float(a[1])
                    newmax=(a0*10**a1)
                    a=newmax/10
                    g('set ztics %s' % str(a))
                    g('set zrange [%g:%g]' % (sizeclass[sc].minFezrange,newmax))
                else:
                    g('set zrange [%g:%g]' % \
                        (sizeclass[sc].minFezrange,sizeclass[sc].maxFezrange))
            g('set term png small crop')
            g('set size 0.8,0.8')
            g('set title "%s"' % title)
            g("set output '%s'" % outfile)
            g("splot '%s' every :%s" % (fname,plot_gap))
            g('set output')
            g('set autoscale z')
            g('set ztics autofreq')
    return flist
#-------------------------------------------------------------
def plot2deps(g,filename,outdir,names,ind,prefix='',graphsList=[],\
        sizeclass=None,Ferange=[]):
    #g=gnuplot handle
    for name in names:
        outfile=prefix+name+'.eps'
        outfile=os.path.join(outdir,outfile)
        graphsList.append(outfile)
        g('set term postscript enhanced eps 17')
        g('set title "%s"' % name)
        g("set output '%s'" % outfile)
        if sizeclass:
            if name=='sulfur': g('set ylabel "Grade(g/g-ore)"')
            if Ferange:
                if name=='ferric' or name=='ferrous':
                    if Ferange[1]<3:
                        a='%e' % Ferange[1];a=a.split('e')
                        a0=ceil(float(a[0]));a1=float(a[1])
                        newmax=(a0*10**a1)
                        a=newmax/10
                        g('set ytics %s' % str(a))
                        g('set yrange [%g:%g]' % (Ferange[0],newmax))
                    else:
                        g('set yrange [%g:%g]' % (Ferange[0],Ferange[1]))
            sizeclassKeys=sizeclass.keys()
            sizeclassKeys.sort()
            #get plot string
            i=2; ss="plot "
            for sc in sizeclassKeys:
                ss+="'%s' using 1:%s title '%s'," %(filename[name].name,\
                    str(i),str(sizeclass[sc].R*10))
                i+=1
            ss=ss.rstrip(',')
            g(ss)
            g('set output')
```

```
            if Ferange:
                g('set autoscale y')
                g('set ytics autofreq')
        else:
            ind+=1
            g("plot '%s' using 1:%s" % (filename,str(ind)))
            g('set output')
            if 'Microb' in filename: ind+=2
    return graphsList
#-------------------------------------------------------
def plot2dpng(g,filename,outdir,names,ind,prefix='',graphsList=[],\
        sizeclass=None,Ferange=[]):
    #g=gnuplt handle
    for name in names:
        outfile=prefix+name+'.png'
        outfile=os.path.join(outdir,outfile)
        graphsList.append(outfile)
        g('set term png small crop')
        g('set size 0.7,0.7')
        g('set title "%s"' % name)
        g("set output '%s'" % outfile)
        if sizeclass:
            if name=='sulfur': g('set ylabel "Grade (g/g-ore)"')
            if Ferange:
                if name=='ferric' or name=='ferrous':
                    if Ferange[1]<3:
                        a='%e' % Ferange[1];a=a.split('e')
                        a0=ceil(float(a[0]));a1=float(a[1])
                        newmax=(a0*10**a1)
                        a=newmax/10
                        g('set ytics %s' % str(a))
                        g('set yrange [%g:%g]' % (Ferange[0],newmax))
                    else:
                        g('set yrange [%g:%g]' % (Ferange[0],Ferange[1]))
            sizeclassKeys=sizeclass.keys()
            sizeclassKeys.sort()
            #get plot string
            i=2; ss="plot "
            for sc in sizeclassKeys:
                ss+="'%s' using 1:%s title '%s'," %(filename[name].name,\
                    str(i),str(sizeclass[sc].R*10))
                i+=1
            ss=ss.rstrip(',')
            g(ss)
            g('set output')
            if Ferange:
                g('set autoscale y')
                g('set ytics autofreq')
        else:
            ind+=1
            g("plot '%s' using 1:%s" % (filename,str(ind)))
            g('set output')
            if 'Microb' in filename and 'Bulk' in filename: ind+=2
```

```
    return graphsList
#-------------------------------------------------------------------
def plot2d(datafiles,phase,speciesNames,microbesNames,fileType,outdir):
    """Plot two-dimensional data (bulk or flowing concentrations)

    phase is a string that holds the name of the phase we are referring to -
    in this case 'bulk' or 'flow'ing phase
    """
    if not(fileType in ['eps','png']) :
        message= "fileType must be eps or png"
        errorHandler(message)
    g=Gnuplot.Gnuplot(debug=0)
    g('unset key')
    g('set data style lines')
    g('set mxtics 5')
    g.xlabel('time(hr)')
    sfname=datafiles['species'].name
    mfname=datafiles['microbes'].name
    graphs=[]

    if fileType=='eps':
        ind=1
        g('set yrange [0:]')
        g('set ylabel "%s" "% ("Conc (g/L " +phase+")"))
        plot2deps(g,sfname,outdir,speciesNames,ind,phase,graphs)
        #plot microbes
        ind=1
        g('set yrange [*:]')
        # g('set yrange [0:]')
        if phase == 'bulk':
            g('set ylabel "Ads+Des Microbes x 10^{10} (cells/L bulk)"')
        elif phase == 'flow':
            g('set ylabel "Microbes x 10^{10} (cells/L flow)"')
        plot2deps(g,mfname,outdir,microbesNames,ind,phase,graphs)
    else:#fileType=='png'
        ind=1
        g('set yrange [0:]')
        g('set ylabel "%s"' % ("Conc (g/L " +phase+")"))
        plot2dpng(g,sfname,outdir,speciesNames,ind,phase,graphs)
        #plot microbes
        ind=1
        g('set yrange [*:]')
        # g('set yrange [0:]')
        if phase == 'bulk':
            g('set ylabel "Ads+Des Microbes x 1e10 (cells/L bulk)"')
        elif phase == 'flow':
            g('set ylabel "Microbes x 1e10 (cells/L flow)"')
        plot2dpng(g,mfname,outdir,microbesNames,ind,phase,graphs)
    return graphs
#-------------------------------------------------------------------
def plot2dAve(ave_datafiles,sizeclass,speciesNames,mineralsNames,fileType,outdir):
    """Plot particle average concentrations
    """
```

```
    if not(fileType in ['eps','png']) :
        message= "fileType must be eps or png"
        errorHandler(message)
    sizeclassNames=sizeclass.keys()
    sizeclassNames.sort()
    #get maximum Ferric & Ferrous values
    smaxFe=[sizeclass[sc].maxFezrange for sc in sizeclassNames]; maxFe=max(smaxFe)
    sminFe=[sizeclass[sc].minFezrange for sc in sizeclassNames]; minFe=min(sminFe)
    g=Gnuplot.Gnuplot(debug=0)
    g('set key outside reverse title "R(mm)" box 3')
    g('set data style lines')
    g('set mxtics 5')
    g.xlabel('time(hr)')
    ave_graphs=[]
    Ferange=[minFe,maxFe]
    if fileType=='eps':
        #plot species
        speciesNames.sort()
        g('set ylabel "Conc x 10^{-3} (g/cm^3 particle)"')
        plot2deps(g,ave_datafiles,outdir,speciesNames,0,'ave',\
            ave_graphs,sizeclass,Ferange)
        #plot minerals
        mineralsNames.sort()
        g('set ylabel "Frac. reacted (%)"')
        plot2deps(g,ave_datafiles,outdir,mineralsNames,0,'ave',\
            ave_graphs,sizeclass)
    elif fileType=='png':
        #plot species
        speciesNames.sort()
        g('set ylabel "Conc x 1e-3 (g/cm^3 particle)"')
        plot2dpng(g,ave_datafiles,outdir,speciesNames,0,'ave',\
            ave_graphs,sizeclass,Ferange)
        #plot minerals
        mineralsNames.sort()
        g('set ylabel "Frac. reacted (%)"')
        plot2dpng(g,ave_datafiles,outdir,mineralsNames,0,'ave',\
            ave_graphs,sizeclass)
    return ave_graphs
#-------------------------------------------------------------
def plotSurf(surf_datafiles,sizeclass,speciesNames,fileType,outdir):
    """Plot particle surface concetrations
    """
    if not(fileType in ['eps','png']) :
        message= "fileType must be eps or png"
        errorHandler(message)
    sizeclassNames=sizeclass.keys()
    g=Gnuplot.Gnuplot(debug=0)
    g('set key outside reverse title "R(mm)" box 3')
    g('set data style lines')
    g('set mxtics 5')
    g.xlabel('time(hr)')
    surf_graphs=[]
    if fileType=='eps':
```

```
            #plot species
            speciesNames.sort()
            g('set ylabel "Conc x 10^{-3} (g/cm^3 particle)"')
            plot2deps(g,surf_datafiles,outdir,speciesNames,0,'surf',\
                surf_graphs,sizeclass)
        elif fileType=='png':
            #plot species
            speciesNames.sort()
            g('set ylabel "Conc x 1e-3 (g/cm^3 particle)"')
            plot2dpng(g,surf_datafiles,outdir,speciesNames,0,'surf',\
                surf_graphs,sizeclass)
        return surf_graphs
    #--------------------------------------------------------------
    def htmlReport(filename,files2d,files3d,simpar):
        """Collect plots in a html document
        """
        #get output directory
        if simpar.plot2d:
            path,dummy=os.path.split(files2d['bulkConc'][0])
        elif simpar.plot3d:
            path,dummy=os.path.split(files3d[0])

        def writePlots(title,fileList):
            html.write("\n<table border=8 cellspacing=3 cellpadding=2>\n\
                <caption><font size=6>%s</font></caption>\n" % title)
            i=0
            files=[os.path.split(ff)[1] for ff in fileList]
            while i<len(files):
                try:
                    html.write('<tr>\n\
                                <td><img src="%s"></td>\n' % files[i])
                    html.write('<td><img src="%s"></td>\n' % files[i+1])
                    html.write('</tr>\n')
                except IndexError:
                    html.write('</tr>\n')
                i+=2
            html.write("</table>\n<br>")

        html=open(os.path.join(path,filename),'w')
        html.write("""<html>
        <head><title>Simulation Results</title></head>
        <body><h1>Simulation Results</h1> """)
        if simpar.plot2d:
            #flowing liquid concentration
            writePlots("Concentration in flowing solution",files2d['flowConc'])
            #Bulk results
            writePlots("Concentration in bulk",files2d['bulkConc'])
            #Surface conc
            if simpar.microbes:
                writePlots("Concentration at the particle surface",\
                    files2d['surfaceConc'])
            #Average files
            writePlots("Average concentration in particle size class",\
```

```
            files2d['avePsolns'])
        #Particle
        if simpar.plot3d:
            writePlots("Concentration in particle size class",files3d)
        html.write("""</table>
            </body>
            </html>""")
        html.close()
#---------------------------------------------------------------
def psReport(filename2d,filename3d,files2d,files3d,simpar):
    """Collect plots in postscript files
    """
    from util import tryCommand
    filename2d=filename2d.split(',')
    def plotFile(mfile,path,fileList):
        mfile=os.path.join(path,mfile) #mfile=merge file
        cmd='epsmerge -o ' + mfile + ' -x 2 -y 3 -par -ps '+ ' '.join(fileList)
        tryCommand(os.system(cmd))
        #remove epsfiles
        for file in fileList:
            os.remove(file)

    #get output directory
    if simpar.plot2d:
        path,dummy=os.path.split(files2d['bulkConc'][0])

        #plot bulkconc graphs
        mfile=filename2d.pop(0)
        plotFile(mfile,path,files2d['bulkConc'])
        #plot flowConc graphs
        mfile=filename2d.pop(0)
        plotFile(mfile,path,files2d['flowConc'])
        #plot Average particle soln graphs
        mfile=filename2d.pop(0)
        plotFile(mfile,path,files2d['avePsolns'])
        if simpar.microbes:
            #plot Surface conc soln graphs
            mfile=filename2d.pop(0)
            plotFile(mfile,path,files2d['surfaceConc'])
    if simpar.plot3d:
        path,dummy=os.path.split(files3d[0])
        plotFile(filename3d,path,files3d)
```

Listing B.5: preamble.py

```
#!/usr/bin/python
"""Executed at the start of each run.

Ensure that all necessary programs/python modules are installed. Otherwise, raise
   error and exit.
"""

import sys,os,shutil
```

```
import modulecheck
from modulecheck import findprograms

##add current directory to path
cwd=os.getcwd()
sys.path.append(cwd)
#required Python modules
modules={
    'Numeric' : 'for numerical computation',
    'scipy' : 'for numerical computation',
    'Gnuplot' : 'curve plotting tool',
    }
#programs that should be installed -- only Gnuplot is essential
programs={
    'wgnuplot' : 'Gnuplot (plotting program)',
    'gswin32' : 'ghostscript,ps/pdf interpreter and previewer',
    'gnuplot' : 'plotting program',
    'gs' : 'ghostscript,ps/pdf interpreter and previewer',
    'epsmerge' : 'a program for merging encapsulated postscript files',
    }
#possible report styles based on programs installed
rstyle={
    'eps' : False,
    'ps' : False,
    'png' : True
    }
#check for installed programs
installed=findprograms(programs.keys())
if sys.platform.startswith('win'):
    if installed['wgnuplot'] is None:
        print "***Gnuplot not installed or not in PATH"
        sys.exit(1)
elif os.name=='posix':
    if installed['gnuplot'] is None:
        print "***Gnuplot must be installed"
        sys.exit(1)

if sys.platform.startswith('win'):
    if installed['gswin32']:
        rstyle['eps']=True
elif os.name=='posix':
    if installed['gs']:
        rstyle['eps']=True
    if installed['epsmerge']:
        rstyle['ps']=True
#check for Python modules
for key in modules.keys():
    if not modulecheck.message(key,msg=modules[key]):
        sys.exit(1)

import time
import string
###import user-defined libraries
```

```
from IO import *
from engine import *

fileType='' #in case a report style is not chosen
```

Listing B.6: modulecheck.py

```
#adapted from
#Python scripting for computational science
#H.P Langtangen
#Springer,2004

def message(module, critical=1, msg=None):
    """
    Import a module and write a message if it is missing.
    critical=0 means that the module is not critical
    (programs may work without). critical=1 means that the
    module must be installed.
    msg is an optional description of the module.
    """
    try:
        exec("import "+ module)
        success = True
    except:
        print "*** The", module, "Python module is not available..."
        success = False
        if msg: print " (%s)" % msg
        if not critical:
            print " ....but this is not critical"
            success = False
    return success

#-----------------------------------------------------
import os,re,sys
def findprograms(programs, searchlibs=[], write_message=False):
    """
    Given a list of programs (programs), find the full path
    of each program and return a dictionary with the program
    name as key and the full path as value. The value is None
    if the program is not found.

    The program list can either be a list/tuple or a
    dictionary (in the latter case, the keys are the programs
    and the values are explanations of the programs).
    If write_message is true, the function writes a message
    if a program is not found. In that case, None is returned
    if not all programs are found.

    A single program can also be given as first argument. In that
    case, findprograms returns True or False according to whether
    the program is found or not.

    Example on usage:
```

```
    if findprograms('plotmtv'):
        os.system('plotmtv ...')

    # write a message if a program is not found:
    if findprograms(['plotmtv'], write_message=True):
        os.system('plotmtv ...')

    programs = ['gs', 'convert']
    path = findprograms(programs)
    if path['gs']:
        os.system('gs ...')
    if path['convert']:
        os.system('convert ...')

    programs = { 'gs' : 'Ghostscript: file format conversions',
                 'convert' : 'File format conversion from ImageMagick',
               }
    if not findprograms(programs, write_message=True):
        print 'the mentioned programs need to be installed'
        sys.exit(1)
    """
    def program_exists(fullpath):
        if sys.platform.startswith('win'):
            # add .exe or .bat to program filename:
            if os.path.isfile(fullpath+'.exe') or \
               os.path.isfile(fullpath+'.bat'):
                return True
        elif os.name == 'posix': # Unix
            if os.path.isfile(fullpath):
                return True
        else:
            raise TypeError, \
                  'platform %s/%s not supported' % \
                  (sys.platform, os.name)
        return False # otherwise

    path = os.environ['PATH'] # /usr/bin:/usr/local/bin:/usr/X11/bin
    paths = re.split(os.pathsep, path)
    fullpaths = {}
    if isinstance(programs, str):
        program = programs
        for dir in paths:
            if os.path.isdir(dir): # skip non-existing directories
                fullpath = os.path.join(dir,program)
                if program_exists(fullpath):
                    return True
        # else, not found:
        if write_message:
            print 'program %s not found' % programs
        return False

    elif isinstance(programs, (list,tuple)):
```

```
        # initialize with None:
        for program in programs: fullpaths[program] = None
        for program in programs:
            for dir in paths:
                if os.path.isdir(dir): # skip non-existing directories
                    fullpath = os.path.join(dir,program)
                    if program_exists(fullpath):
                        fullpaths[program] = fullpath
                        break # stop when the program is found

    elif isinstance(programs, dict):
        # initialize with None:
        for program in programs.keys(): fullpaths[program] = None
        for program in programs.keys():
            for dir in paths:
                if os.path.isdir(dir): # skip non-existing directories
                    fullpath = os.path.join(dir,program)
                    if program_exists(fullpath):
                        fullpaths[program] = fullpath
                        break

    if write_message:
        missing = False
        for program in fullpaths.keys():
            if not fullpaths[program]:
                if isinstance(program, dict):
                    print "program '%s' (%s) not found" % \
                          (program,programs[program])
                else  # list or tuple
                    print 'program "%s" not found' % program
                missing = True
        if missing:
            return None

    return fullpaths
```

B.2 Sample Input Files – Baseline

This section contains sample input files to the program. The input files are plain text. The definition of each entity in the input files starts at a 'begin' keyword followed by the name of the entity. and stops at the 'end' keyword.

Listing B.7: User defined parameters

```
#Simulation parameters
#NOTE:# preceeds comments
#Please DO NOT change order of parameters.
begin simpar
#****PHYSICAL PARAMETERS****
#ore density(g/cm^3
ore_density= 1.45
#particle porosity
porosity= 0.08
#Stagnant liquid volume fraction
stagVolFrac= 0.3
#Flowing liquid volume fraction
flowVolFrac= 0.1
#Agglomerate temperature (degree C)
temperature=38.6
#length of the unit volume (Unit volume is a cube) (cm)
L=20
#superficial velocity of flowing liquid (cm/min)
uf=0.05
#---Mass transfer coefficients---#
#Mass transfer coefficient at particle surface/bulk interface (cm/min)
ksb=0.1
#Mass transfer rate at flowing/stagnant liquid interface (1/min)
#kbf = mass transfer parameter*[exchange area/total volume]
kbf=0.1
#****SIMULATION PARAMETERS****
#Simulation time(hrs)
simTime=8400
#Number of timesteps to take
timesteps=5000
#plot gap - plot ev ry nth solution
plot_gap=150
#report style - eps, postscript or html. leave blank if no report is wanted
report_style =html
#name of output directory - in the "simulation--results" directory
dirname = base
#BOOLEAN VALUES (yes/no). Default=yes
#graphs to plot
plot2d=yes
plot3d=yes
#concentrations to set constant in the bulk
constant ferric =no
constant ferrous=no
constant acid =no
```

```
constant oxygen =no
#biotic or abiotic system.
microbes=yes
end
```

Listing B.8: Mineral parameters

```
#Mineral parameters
#Please DO NOT change order of parameters or minerals.
# NOTE:# precedes comments
#
#The grade is a number or comma separated list. Each entry in the list corresponds to the grade of
#the mineral in the each particle size class defined. The grades in the list MUST be in the order of
#the particle size classes in the psc_par file.
#
#Mineral type — sulfide or oxide
#Stoichiometry columns— Cu, Fe3+, Fe2+,Acid (*for reactions taking place inside the particles*)
begin chalcocite
                type = sulfide
(*mol/g ore*)   grade = 5.66e-5,5.66e-5,5.66e-5
(*g/mol*)       molar_mass=159.14
                Topological_Exponent(*phi*)= 1.3
(*J/mol*)       Activation_Energy= 23900
(*1/min*)       Rate_Constant= .446
(*degree C*)    Reference_Temperature= 35
                stoichiometry=0.8,-1.6,1.6,0
                Electrochemical_Exponent(*m*)= 0.124
(*mol/L*)       Ka= 0.154
(*mol/L*)       Kb= 0.00001
                sulfur_stoic=0.
end
#
begin covellite
                type = sulfide
(*mol/g ore*)   grade = 5.66e-5,5.66e-5,5.66e-5
(*g/mol*)       molar_mass=541.54
                Topological_exponent(*phi*)= .6
(*J/mol*)       Activation_Energy= 97900
(*1/min*)       Rate_Constant= .01
(*degree C*)    Reference_Temperature= 75
                stoichiometry=1.2,-2.4,2.4,0
                Electrochemical_exponent(*m*)= .5
(*mol/L*)       Ka= 0.0147
(*mol/L*)       Kb= 0.00001
                sulfur_stoic=1.
end
#
begin pyrite
                type = sulfide
(*mol/g ore*)   grade = 0.0003,0.0003,0.0003
(*g/mol*)       molar_mass=119.97
                Topological_exponent(*phi*)= 2.
(*J/mol*)       Activation_Energy= 74300
```

```
(*1/min*) Rate_Constant= .00005
(*degree C*) Reference_Temperature= 55
             stoichiometry=0., -(2+6*(2-beta)), 3+6*(2-beta), 4*(2-beta)
             Electrochemical_exponent(*m*)= .5
(*mol/L*) Ka= 0.00001
(*mol/L*) Kb= 0.00001
(*between 0-1*) sulfur_stoic=0.
end
```

Listing B.9: Species parameters

```
#Species parameters
#NOTE: a # precedes comments
#Please DO NOT change order of parameters.
begin copper
(*g/L*) init_particle_conc= 0.0
(*g/L*) init_bulk_conc= 0
(*g/L*) init_inletflow_conc=0
(*m^2/s*) diffusivity= 1e-9
(*g/mol*) molar_mass=63.54
end
#
begin ferric
(*g/L*) init_particle_conc= 0.0
(*g/L*) init_bulk_conc= .1
(*g/L*) init_inletflow_conc=.1
(*m^2/s*) diffusivity= 1e-9
(*g/mol*) molar_mass=55.85
end
#
begin ferrous
(*g/L*) init_particle_conc= 0.0
(*g/L*) init_bulk_conc= .1
(*g/L*) init_inletflow_conc=.1
(*m^2/s*) diffusivity= 1e-9
(*g/mol*) molar_mass=55.85
(*J/mol*) Activation_Energy= 68600
(*1/min*) Rate_Constant= 31.17
(*degree C*) Reference_Temperature= 100
end
#
begin acid
(*g/L*) init_particle_conc= 0.0
(*g/L*) init_bulk_conc= 8
(*g/L*) init_inletflow_conc=8
(*m^2/s*) diffusivity= 3e-9
(*g/mol*) molar_mass=98.07
(*J/mol*) Activation_Energy= 20000
(*1/min*) Rate_Constant= 9.3e-4
(*degree C*) Reference_Temperature= 20
end
#
begin oxygen
```

```
(*g/L*) init_particle_conc= 0.
(*g/L*) init_bulk_conc= 8e-3
(*g/L*) init_inletflow_conc=8e-3
(*m^2/s*) diffusivity= 1e-9
(*g/mol*) molar_mass=32
end
```

Listing B.10: Microbial parameters

```
#Microbial parameters
#NOTE: a # precedes comments
#Please DO NOT change order of parameters.
#oxidizes -- iron , sulfur
#type -- mesophile,thermophile
begin FeOx
      oxidizes = iron
(*cell/L*) init_desorbed_conc=1e10
(*cell/L*) init_inletflow_conc=1e10
#*******physical parameters
(*g/mol*) molar_mass=1e12
(*m^2/s*) diffusivity= 5e-9
(*cell/mol Fe2+*) yield=2e12
(*mol Fe2+/cell-min*) maintenance_factor=0
(*1/min*)   max_growth_rate=0.0017
(*1/min*)   death_rate =0.00017
#*******adsorption parameters(*Langmuir*)
(*cells/*g* ore*) adsorption_max=1.5e9
(*L H20/cell*) adsorption_constant=67e-12
#******temperature parameters(*Ratkowsky*)
(*degree C*) Tmin=9.6
(*degree C*) Tmax=49.5
(*degree C*) Topt=38.6
      b=0.01551
      c=0.22061
#******monod parameters
(*mol/L*) inhibition_factor=1e12
(*mol/L*) oxygen_monod_factor=.00005
(*mol/L*) ferrous_monod_factor=.0001
(*mol/L*) acidlim_factor=.01
end
#
begin SuOx
      oxidizes = sulfur
(*cell/cm^3 agg*) init_agg_conc=1e12
(*cell/L*) init_inletflow_conc=1e12
#*******physical parameters
(*g/mol*)   molar_mass=1e12
(*m^2/s*)   diffusivity= 5e-9
(*cell/mol S*) yield=2e11
(*mol S/cell-min*) maintenance_factor=0
(*1/min*)   max_growth_rate=0.00083
(*1/min*)   death_rate =0.000083
#*******adsorption parameters(*Langmuir*)
```

```
(*cell/*g* ore*)  adsorption_max=1.5e9
(*L H20/cell*)  adsorption_constant=17e-12
#******temperature parameters(*Ratkowsky*)
(*K*)   Tmin=-4.2
(*K*)   Tmax=39.7
(*K*)   Topt=32.8
        b=0.00293
        c=0.30891
#*******monod parameters
(*mol/L*)  inhibition_factor=1e12
(*mol/L*)  oxygen_monod_factor=.00005
(*g/g*)   sulfur_monod_factor=.00192
end
```

Listing B.11: Size class parameters

```
#Particle size class parameters
#NOTE: a # precedes comments
#Please DO NOT change order of parameters.
#Add numbers in ascending order to the end of a class name to differentiate between the classes.
begin psc1
        grid_points=25
(*mm*) radius= 1.0
        massfrac=.334
end
begin psc2
        grid_points=30
(*mm*) radius=3.0
        massfrac=.333
end
begin psc3
        grid_points=30
(*mm*) radius=5.
        massfrac=.333
end
```