GEOMETRIC METHODS FOR OPTIMIZING RAMIFIED FLOW NETWORKS

BY

MARTIN SCOTT SINGLETON

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Physics in the Graduate College of the University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Professor Yoshi Oono, Chair
Associate Professor Alfred Hubler, Director of Research
Professor Mike Weissman
Professor John Stack
Abstract

Using geometrically based approaches, optimal characteristics of nonlinear flow networks were examined. The first model studied was that of a fractal-like desalination plant. Solving the equations of diffusion by analogy to electrostatics, desalination graphs superimposed on the diffusion gradients were optimized to produce maximal water flow. Graphs were generated while varying branching angles and ratios to determine optimal morphologies. Three different boundary conditions are discussed, those of constant pressure difference, constant water flow rate, and constant absorber salinity. Another type of flow network, that of the perceptron, was analyzed from the point of view of its attractors in order to determine which patterns resonated with the network. Using the definition of resonance as learning rapidity, convergence rates were studied by calculating the mean paths to convergence for perceptron weights. Polynomial expressions were given for the mean and variance steps to convergence, which depended upon two dimensionless geometric parameters.
To Father and Mother, may they rest in peace. To my dear wife Kyram.
Thanks to my advisor and collaborator Alfred Hubler. Thanks to Gregor Heiss for his collaboration on part of this project. Thanks to Rhanor Gillette for his encouragement and support. Thanks to my wife Kyram R. Singleton for her assistance on the figures, as well as her wonderful love and companionship. Thanks to my mother and father for always being there for me when they could. Thanks to my sisters and brothers Lisa, Gordon, Nancy, Peter, and Pam, who many times listened patiently to me during this long journey and rendered thoughtful advice. Thanks to Aldo Senior and Junior, Ingrid, and Melissa. Special thanks to my old college buddy, Rudy Covarrubias, for his wise council and very enthusiastic encouragement. Thanks to my childhood friend Mathew Ogburn for his encouragement. Thanks to Peter Fleck, Glenn Foster, Davit Sivil, David Lyon, Onyeama Osuagwu, and Jeremy Tan for many interesting and stimulating conversations. Thanks to the one thousand or so very bright undergraduate students whom I had the privilege and opportunity to teach and learn from during my time at the University of Illinois. Thanks to my committee for their participation and wisdom. Thanks to the UIUC Physics department for its financial support and opportunity to be in the program.
# Table of Contents

List of Tables ........................................ vii
List of Figures .................................... viii
List of Symbols .................................... xi

Chapter 1  Introduction .............................. 1

Chapter 2  Optimization of Ramified Absorber Networks Doing Desalination .................................. 9
  2.1  Introduction .................................. 10
  2.2  Physical Situation of the Posed Problem .............. 12
  2.3  Ramified Graphs and Iterated Function Systems ......... 14
  2.4  Spatial dependence of the salt concentration .......... 15
  2.5  Discussion .................................. 19
  2.6  Conclusion .................................. 28

Chapter 3  Further Investigation of Optimal Desalination Graphs 32
  3.1  Other Boundary Conditions for Model of Chapter 2 ....... 32
    3.1.1 Constant Salinity Absorbers ..................... 32
    3.1.2 Constant Water Product Absorbers .............. 34
  3.2  Further Investigations .......................... 37
    3.2.1 Optimal Graphs by Optimizing All Angles, Ratios, and Weights .................. 37
    3.2.2 Self-Adjusting Graphs ......................... 38
    3.2.3 3-D Graphs ................................. 39
    3.2.4 Experiments on Desalination .................... 40

Chapter 4  Learning rate and attractor size of the single-layer perceptron .................................. 42
  4.1  Abstract .................................. 42
  4.2  Introduction .................................. 42
  4.3  Perceptron Algorithm ........................... 44
  4.4  Convergence Rates ............................. 51
  4.5  Results .................................... 54
  4.6  Discussion .................................. 56
  4.7  Conclusion .................................. 58

Chapter 5  Conclusions ............................... 60

Appendix A  Iterated Function System Linear Transformation 61

Appendix B  Expansion about $c_i' = \frac{\Delta}{RT}$ ............... 62
List of Tables

2.1 Typical desalination values used to solve Eq. (2.3) .................. 19

4.1 Inequalities corresponding to attractors of Fig. (4.1), for the representatives of the five function classes. In Fig. (4.1) the threshold is set to $-t = -1$ .......................... 51

4.2 Predicted and observed means and standard deviations of the number of steps to converge from Eqs. (4.16-4.25) and 10,000 trials. To compare note that e.g. the standard error of the mean for (0,0,0,0) would be $23.98/\sqrt{10000} = .24$ .......................... 54

E.1 Details utilized to determine all possible trajectories for weight vectors initially in the different regions $A_1 - A_7$. The condition violated was used to select the weight change, which in turn gave the slope of the trajectory segment. By determining the distance $d_R$ of the complete trajectory from arbitrary starting location to the attractor $A_6$, and multiplying by the rate $r_R$ for each different segment, the integral of Eq. (E.1) was written down .................. 74
List of Figures

2.1 Ramified graph which is optimal for (i.e. it maximizes) water production occurs for $G = 7$, $r = .521$, and $\alpha = 1.16$. The graph is optimal in the sense that other graphs with the same length and total area of absorbers, but with different branching angles and branching ratios, have lower water production rates. . . . . . . 13

2.2 Schematic for a ramified network consisting of pipes with diameters $w_{g,i}$, where the angles, ratios, and diameters depend on the generation $g$. Red shading indicates effective membrane surface. For a network of total length $l$, $U_{g,i}$ and $V_{g,i}$ denote $l u_{g,i}$ and $l u_{g,i}$ respectively. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3 Ratio for ramified graph networks having maximal water output for the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. Ratios were deemed optimal if the corresponding water production rate was maximal over all angles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

2.4 Angle for ramified graph networks having maximal water output for the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. Angles were deemed optimal if the corresponding water production rate was maximal over all ratios. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

2.5 Optimal water production rates in $m^3/s$ for ramified graph networks in the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. These are the values of water production per unit time that were maxima (and hence optimal) for each generation by independently varying ratios and angles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

2.6 Ramified graphs having optimal properties (maximal water production rate for constant pressure difference absorbers) as in Figs. 2.3-2.4 for generations $G = 2$ to $G = 10$. The graphs are optimal in the sense that other graphs with the same length and total area of absorbers, but with different branching angles and branching ratios, have lower water production rates. . . . . . . . 21

2.7 Contours of total water production for constant pressure difference absorbers over the ramified graph network for equal increments of angles between 0 and $\pi$, and ratios of 0 to .9, for generations $G = 3$, 5, and 7, with parameter values as in Table 2.1. Lines of singularity occur where the coefficient matrices were non-invertible. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
2.8 Plots of water production $W$ versus ratio $r$ with $G = 7$ and $\alpha = 1.167$ (top), and water production $W$ versus angle $\alpha$ with $G = 7$ and $r = .5155$ (bottom).  
2.9 Optimal water production rate in m$^3$/s for $G = 7$ as a function of the energy consumption per volume $E$ of permeate produced.  
2.10 Optimal water production for $G = 3$ versus ratio difference (top) and angle difference (bottom). For the ratio difference plot, alpha was the optimal angle, and ratios were varied independently about the optimal ratio. For the angle difference plot, ratio was the optimal ratio, and angles were varied independently about the optimal angle.  
2.11 Curve fits of optimal water production while varying independent dimensionless parameters. On the top is shown the effect of varying $\beta$, while the effect of varying $k$ is shown at bottom. All curves were fit to the general parabolic form of $W(G) = C_1 - C_2 (G - G_{max})^2$.  
2.12 Dependence of $G_{max}$ from Eq. (2.21) on the dimensionless parameters $\beta$ (top) and $k$ (bottom).  
3.1 Total water production for constant salinity absorbers over the ramified graph network for equal increments of angles between 0 and $\pi$, and ratios of 0 to .9, for generations $g = 2$ to $g = 10$. Note that since the boundary conditions are different in this case as for those for Fig. 2.7, the fine structures (lines of singularities) of the plots are largely non-existent.  
3.2 Angle (blue/dashed) and ratio (red/solid) for ramified graph networks having maximal water output for the case of constant salinity absorbers as function of number of generations, for generations $g = 2$ to $g = 10$.  
3.3 Ramified graphs having optimal properties (maximal water production rate for constant salinity absorbers) for generations $g = 2$ to $g = 10$.  
3.4 Schematic for a ramified graph consisting of pipes where the angles, ratios, and diameters depend on the generation $g$.  
3.5 The red mangrove, a plant with a fractal root structure which lives in coastal habitats that does reverse osmosis.  
3.6 Concentration cell for efficient desalination using a fractal electrode and electrolysis, with $H^2$ recycling.  
4.1 The solutions of the linear inequalities in Table 4.1, corresponding to the attractors of the five learnable classes for the single-layer perceptron. a) Class I representative $(0,0,0,0)$, b) Class II representative $(0,0,0,1)$, c) Class III representative $(0,0,1,0)$, d) Class IV representative $(0,0,1,1)$, e) Class V representative $(0,1,1,1)$. Note that these regions coincide exactly with the attractors in weight space shown in Fig. (4.2).
4.2 The paths in weight space for 250 trials of the converging weight vectors. x: Beginning point of trajectory. *: End point of trajectory. 

a) Class I representative (0,0,0,0), \( \mu = 24.47, \sigma = 22.19 \),  
b) Class II representative (0,0,0,1), \( \mu = 76.33, \sigma = 40.80 \),  
c) Class III representative (0,0,1,0), \( \mu = 43.59, \sigma = 34.34 \),  
d) Class IV representative (0,0,1,1), \( \mu = 41.44, \sigma = 31.04 \),  
e) Class V representative (0,1,1,1), \( \mu = 37.91, \sigma = 27.04 \). By calculating the mean and standard deviation of the number of steps over these paths, we arrived at expressions 4.16-4.25.

4.3 The histograms of the five different classes for 10,000 trials are shown. a) Class I representative (0,0,0,0), \( \mu = 25.77, \sigma = 23.98 \),  
b) Class II representative (0,0,0,1), \( \mu = 77.50, \sigma = 40.06 \),  
c) Class III representative (0,0,1,0), \( \mu = 44.39, \sigma = 35.58 \),  
d) Class IV representative (0,0,1,1), \( \mu = 40.24, \sigma = 32.45 \),  
e) Class V representative (0,1,1,1), \( \mu = 37.05, \sigma = 29.22 \). Note that four classes have fairly similar shapes, having their distributions weighted towards the left, while the 'and' rule is centrally distributed. Trials which were already in the attractors at \( n = 0 \) were omitted for clarity.

E.1 Regions \( A_1 - A_7 \) used for calculating Eq. (4.20) from Eqs. (4.10) and (4.12). The dashed lines corresponding to \( w_1 = a, w_2 = a, \) and \( w_2 = -w_1 + a \) partition the initial weight vector space into the seven regions \( A_1 - A_7 \).
List of Symbols

\( \mu \) viscosity of the medium in kg m\(^{-1}\)s\(^{-1}\)

\( k \) dimensionless membrane resistance

\( \beta \) dimensionless inverse svelteness ratio

\( G \) total number of generations

\( r \) branching ratio

\( r_{L,R} \) left and right branching ratio

\( r_R \) rate equal to the number of steps per unit distance travelled by the weight vector

\( \phi \) iterated function system

\( c \) salt concentration in mol/m\(^3\)

\( J \) flux of \( c \)

\( D \) diffusion coefficient in m\(^2\)/s

\( \Delta_i \) applied pressure drop at each absorber membrane

\( \Delta_m \) membrane flow resistance

\( c_0 \) salt concentration in mol/m\(^3\) inside the absorber

\( c_\infty \) salt concentration in mol/m\(^3\) at infinity

\( W_i \) water production rate

\( \kappa \) membrane permeability in m\(^2\)

\( b \) thickness of the membrane

\( R \) ideal gas constant in J/Kmol

\( T \) temperature in Kelvin

\( Q_i \) salt production rate of node \( i \)

\( E_i \) energy consumption

\( \tilde{c}_i \) dimensionless concentration

\( c_\delta \) difference in salt concentration between the inside of the absorber and infinity
ξ normalized applied pressure
α_L left branching angle
α_R right branching angle
L total length of network
⌈⌉ ceiling function
σ^2 variance
µ_k expected number of weight changes to converge
⊕ tensor addition
ω^n weight vector
d_j desired output for input vector x_j
y actual output of neuron y
Θ Heaviside function
<w_{i,n}> components of average trajectory
<w^k>_R range of weight vectors within region R which produce the same output for y
d_R distance computed from the direction C_R of the trajectory in R and the location of the next region
C_R piecewise constant vector field
P number of patterns
e adaptation
Z set of integers
R set of real numbers
Z^+ set of positive integers
∇· the divergence of a three dimensional vector field
∇ the gradient of a scalar function
∇^2 the Laplacian of a scalar function
h_{L,R} linear transformations which represent an iterated system of functions
A_G total length of the network after G generations
A_{g,i} line segment u_{g,i}v_{g,i} between nodes u_{g,i} and v_{g,i}
A_{1,1} stem segment
S = {V_{G,i}} locations of absorbers
U_{1,1} outlet of the permeate
R_a radius of spherical absorber
r = (x, y) location of measurement of the concentration c
Ω spherical region of radius r centered at a single salt absorber of radius r > R_a
Chapter 1

Introduction

The use of geometric techniques and methods to solve mathematical problems can often lead to elegant solutions. In the modern sciences, such approaches have not enjoyed widespread application until relatively recently. Indeed, during most of the twentieth century mainstream physics dealt with elaborating results in quantum physics and statistical mechanics, using tools of applied analysis such as the Taylor series expansion. In the study of fields, tensor theory has been quite useful. In modern high energy theoretical science, perturbation methods are predominant. Yet despite such high-powered mathematical techniques, the ability to understand the qualitative behavior of the solutions to such equations often seems to lag their sophistication. Moreover, the advent of ever more powerful digital computers can place a higher premium on simpler approaches and models which lend themselves more to computation. In particular, the methods of nonlinear dynamics, such as chaos and fractal theory, are becoming increasingly relevant.

Fractal shapes [40], whose morphologies are self-similar at all scales, have their most direct origin in exemplars such as the Koch snowflake and the Sierpinski triangle described initially in the early 20th century. Very recently researchers have begun to study systems composed of fractal components, as opposed to the conventionally studied shapes such as spheres, cylinders, and planes. One application of this idea is to the study of networks which optimize thermal conductivity. Borrowing from the natural tree-like forms found in nature, Yu and Li [83] examined the properties of networks composed of H-shaped fractals with fixed area and length ratios, $\beta$ and $\gamma$ respectively, in order to reduce thermal conductivity. Using a thermal-electric analogy, they calculated the thermal conductivity of the fractal network. I.e., each branch of
the network can be thought of as a resistor, so that using the parallel and series equivalent resistances, the equivalent resistance of the network (and hence the conductance) was calculated. These conductivities depend only on $\gamma$, $\beta$, and the cross-sectional area and length of the first branch. Dimensionless effective conductivities for composite networks consisting of the fractal network embedded into a matrix were then calculated. By comparing the effective conductivities of the composite fractal network to those of the underlying material, it was found that the conductivities of the fractal composites are orders of magnitude less when the branches became slenderer and the network denser.

In a similar study, Chen, Yu, Xu, and Li [16] studied the permeability of composites of V-shaped fractal-like tree flow networks. The permeability of a single channel was obtained using Darcy’s law for the permeability in terms of the flow rate, viscosity, pressure difference, and cross-sectional area. Then, for small Reynold’s numbers the pressure difference is given by the Hagen-Poiseuille equation, which also depends on the flow rates. In this way by combining the two equations the flow rate was eliminated and expressions for the permeability were found which depend only on the ratio of channel lengths $\gamma$, the ratio of channel diameters $\beta$, and the original channel length and diameter. Assuming an anisotropic, porous medium, the components of the permeability tensor were used to derive dimensionless effective permeabilities for the $x$- and $y$-directions of the network embedded in a permeable matrix. Their results showed that the effective permeability of the fractal composite networks is greater than that of the traditional parallel-channel networks for values of $\beta$ above a certain threshold, which depends also on $\gamma$, the branching angle, the relative surface porosity, and the iteration number.

These findings illustrate a deeper underlying physical principle that has come to be known as the constructal theory [8]. In configurational non-equilibrium thermodynamics, the constructal theory says that flow structures which aren’t in equilibrium will arrive at equilibrium configurations by optimizing their freedom to morph, their performance $R$, and their svelteness $S_V$ for fixed external size.
where the svelteness is given by

\[ S_V = \frac{L}{V^{1/3}} \]  

(1.1)

and \( V \) is the internal volume of the flow configuration. Thus for a flow system to survive, it must evolve towards minimizing the resistance to the flow. This principle has been found to apply to inanimate flow configurations such as duct cross sections, open channel cross sections, tree-shaped fluid flow river basins, turbulent flow structures, dendritic crystals, and the global air circulation. Animate configurations which exhibit the constructal theory do so with respect to processes and characteristics such as body heat versus body size, breathing and heartbeating, flying, running, swimming, and organ size. Recently this constructal law has also found its way into many engineering applications such as in the design of flow spacings, trees for heat conduction and fluid flow, and multiobjective flow architectures.

An earlier attempt at characterizing optimal flow networks, known as Murray’s law [48], gave a relation between the parent and daughter branches of a tree-like vascular system, such as of the lungs. Murray, assuming that Hagen-Poiseuille dynamics prevails, optimized the network to minimize the energy consumption, finding that

\[ r_p^3 = r_{d_1}^3 + r_{d_2}^3 + \ldots + r_{d_n}^3, \]  

(1.2)

where \( r_p \) is the radius of the parent branch and \( r_{d_i} \) are the radii of the daughter branches. The optimal fractal networks which were studied by Chen et. al. did not obey Murray’s law [16]. However it is also possible that the assumption of Hagen-Poiseuille dynamics may not be sufficient to describe the networks which Murray studied. Although Murray applied his considerations to capillary trees, a more recent study of fluid flow in the bronchial tree by Mauroy, Filoche, Andrade, and Sapoval [41] could shed light on the issue. The use of Darcy’s law to describe flow through branched structures yields a linear relation between the flow and the pressure drop, since a low Reynolds number is assumed. Based upon such an assumption, models of porous media with a network of bifurcating
branches typically predict only uniform and synchronous flow distributions in airways. Yet if inertial effects dominate over viscosity effects, then a more appropriate approach may be to study numerically the system using the Navier-Stokes equation. Based on such assumptions, Mauroy et. al. showed that for higher Reynolds numbers, a bifurcating tree-like structure with variable rotation angle $\alpha$ of the branched bifurcations and various fixed length to diameter ratios, large flow asymmetries resulted as $|\alpha|$ increased from zero (i.e. from the planar case). In addition, they showed that the flow distribution depends strongly on the Reynold’s number and on the aspect ratio $L/D$. This would suggest for instance that the flow distributions for rest and exercising should differ markedly.

A useful application of the notion of optimal configurations for flow networks is to the challenge of desalination. Indeed, it has been estimated recently that over 1 billion of the earth’s population lack access to safe drinking water [65]. The problem of fresh water availability is becoming increasingly more difficult in a world of rapidly growing population. One complicating factor is that over 80% of all fresh water consumption goes towards agricultural and energy usage [65]. Another difficulty is that – possibly due to the earth’s climate changes – glaciers, and along with them the earth’s major rivers, have been receding.

Commercially the most important desalination technology is reverse osmosis, which filters water through a (typically) polyamide membrane by applying hydraulic pressure $p$ to overcome the osmotic pressure $\Pi = cRT$ of the dissolved salts, where $c$ is the salt concentration in molar, $R = 8.314 \text{J/mole/K}$ is the ideal gas constant, and $T$ is the temperature in Kelvins. In recent years reverse osmosis has largely superseded thermal distillation, which is less thermodynamically efficient. Of notable new desalination technologies is included forward osmosis, which counteracts the osmotic pressure in the saline solution with a greater osmotic pressure of dissolved ammonia and carbon dioxide in the desalted solution [22]. Another new method of desalination being introduced is known as low-temperature thermal distillation [9] (LTTD). Based upon an idea originally attributed to d’Arsonval, who sought to apply it in the generation of electrical power, LTDD pumps cold water from the ocean’s depths to the surface where the vapors of the warmer surface water are thereby condensed to produce
fresh water. Each of these desalination technologies, as well as the others, suffer from at least one prominent drawback. Although reverse osmosis for example is generally one of the more energy efficient methods, its chief byproduct concentrated brine is harmful for the environment. The thermodynamic limit on efficiency for any desalination method is embodied in a reversible process [23]. Consider the case of reverse osmosis: the minimal applied pressure \( p \) to produce pure water will be the osmotic pressure \( \Pi \). The minimal work required to move a volume of water over one meter through a square meter filter will be

\[
\int \mathbf{F} \cdot \mathbf{dx} = cRT = 0.76\text{kWh},
\]

using typical values from Table 2.1. Thus 0.76kWh is the theoretical minimum amount of energy that must be input in order to produce one cubic meter of pure water from sea water. The most efficient reported large-scale desalination plant (which happens to employ reverse osmosis) expends 1.58kWh/m\(^3\) [60]. Hence it seems worthwhile, given also its great social and economic importance, to attempt further theoretical improvement of desalination technologies.

Methods of self-adjustment should also be investigated theoretically in order to provide complimentary technologies to those already being considered. In recent times the notion of smart materials has been of considerable interest. One example of this is the application of information technology and materials science advances towards the development of a self-healing electronic power grid [2]. Along these lines are systems which self-assemble. For instance researchers have recently created self-assembling protein fibers which through rational mutations can be altered with respect to assembly, stability, and morphology [49].

A specific system showing self-assembly which has been well-studied is that of a self-agglomerating set of metallic particles between two electrodes. In 1996, Dueweke, Dierker, and Hubler [21] showed that based on the principle of minimal resistance, the speed of self-assembly was a linear function of the initial particle spacing and inter-electrode spacing, with the assembled patterns showing the self-healing property under small perturbations. In 2004, Smyth and Hubler created a dynamical model which, assuming a principle of minimum dissipation
per channel, showed a phase transition between closed-loop and open-loop dendritic networks, in order to explain not only the self-agglomeration of metallic particles between two electrodes but the natural formation of river basin flow networks [67]. This research was further extended in 2005 when Jun and Hubler [32] delineated three distinct stages of growth of the networks, strand formation, boundary formation, and then geometric expansion, each of which was characterized by its own characteristic process: strand formation by cooperative movement of particles towards the outer boundary, boundary formation by the rapid connection of particles towards the outer boundary and to the boundary, and geometric expansion by the particles filling the available space while maintaining the network topology.

Given that these particles interact and form dendritic structures, it seemed natural that they too exhibited a type of learning. Indeed, in 1999 Sperl, Chang, Weber, and Hubler had already shown that self-agglomerating conducting particles in a dielectric medium exhibited Hebbian learning of the history of the system which was represented by the strength of connection between different electrodes [68]. The field of artificial intelligence provides possible mechanisms for self-adjustment of flow networks in the form of neural networks [6]. Although the study of neural networks has its roots in the 1940’s with the publications of McCulloch and Pitts on so-called neuro-logical networks [44], the field really began to gain momentum with the introduction by Rosenblatt in 1962 of his extensive empirical and theoretical studies of the perceptron [58]. Although the perceptron was originally utilized to model the function of retinal cells, it has also been relevant in studying Purkinje cells, found in the cerebellum, a part of the brain thought to be associated with the modulation of motor commands and possibly language learning [37]. In fact, recently Brunel and collaborators, establishing that optimally reliable learning took place for perceptron networks with at least 50% silent synapses, fitted the optimal perceptron weight distribution to that of the synapses between a purkinje cell and granular cells. The close correspondence suggested an information storage capacity for a single Purkinje cell of 4 Kilobytes of information in the form of 40,000 input-output connections, which indicates a total information capacity of the Purkinje cells at 600
Gigabits of information, taking into account the estimate of 15 million Purkinje cells in the cerebellum [66]. Despite their wide-ranging applicability in both artificial and as models of biological neural networks, the feedforward perceptron networks do not display true Hebbian (associative) learning. In 1982 Hopfield [29] introduced an artificial neural network model consisting of nonlinear, graded response neurons which were organized into symmetric networks having associative memory [30]. This type of neural network was especially noteworthy as its convergence criterion was the minimization of a network energy $E$, given by

$$E = -\frac{1}{2} \sum_{i,j} J_{ij} S_i S_j - \sum_j I_j S_j,$$

(1.4)

where $J_{ij}$ are the connection strengths (weights) between neuron $i$ and neuron $j$, $S_j$ is the activity of neuron $j$, and $I_j$ is the external input into neuron $j$.

This key insight led to many later developments in the statistical mechanics of neural networks, including Bayesian techniques, as well as the study of learning as a stochastic process, in which the algorithms generate processes which resemble the dynamics of physical systems with an associated energy $E(S)$ [79]. One such development was the application of replica theory to the Hopfield network by Gardner in 1988 [25]. In this model, neurons were treated as $N$ Ising spins $S_i$, which would learn $p = \alpha N$ $N$-bit spin configurations or patterns, $\xi^\mu_i = \pm 1$, $\mu = 1, \ldots, p$; $i = 1, \ldots, N$. A fractional volume $V_T$ which is in effect a partition function of the system is written down, for a given realization of the random patterns. Then, since it can be seen that $V_T = \prod_{i=1}^N V_i$, where the $V_i$ are the fractional volumes in the spaces of interactions $\{J_{ij}\}$ for fixed $i$, in the thermodynamic limit one has to study

$$\lim_{n \rightarrow \infty} \frac{1}{N} \ln V_T = \frac{1}{N} \sum_i \ln V_i.$$

(1.5)

Assuming $V_T$ is self-averaging, one need only calculate $<\ln V>$, the average of $\ln V_i$ over the quenched distribution of the patterns $\xi^\mu_i$, which is accomplished by the so-called replica trick:

$$<\ln V> = \lim_{n \rightarrow 0} \frac{<V^n>-1}{n}.$$

(1.6)
Gardner went on to show that with the volume (partition function) depending on the storage ratio $\alpha$, the product $\kappa$ of the spin and the magnetic field at each site, and the magnetization $m$, (correlation between patterns) the capacity increases with $m$ from $\alpha = 2$ with $\kappa = 0$, tending to infinity as $m$ goes to 1. Subsequently, the replica method, as well as another equivalent mean field theory approach known as the cavity method, was also used to study perceptron networks [[26], [13]].

In the foregoing, many complexities in the existing literature and methodology have been addressed, as a way of setting the backdrop for what follows. In the following are presented physically motivated, geometric approaches to the questions considered above. Chapter 2 gives an optimization study for ramified flow networks embedded in a diffusive network. Here the application is to desalination. Chapter 3 discusses some possible extensions of these ideas. Chapter 4 presents a possible candidate for creating self-adjusting desalination networks, the perceptron. The perceptron study focuses on questions of speed of convergence and resonant learnable patterns. Appendices F and G contain Matlab code used to perform these studies.
Chapter 2

Optimization of Ramified Absorber Networks Doing Desalination

Reprinted with permission from: Martin S. Singleton, Gregor Heiss, and Alfred Hubler, Physical Review E, EQ10558 accepted for publication, 2010. Copyright 2010 by the American Physical Society.
Abstract

An iterated function system is used to generate fractal-like ramified graph networks of absorbers, which are optimized for desalination performance. The diffusion equation is solved for the boundary case of constant pressure difference at the absorbers and a constant ambient salt concentration far from the absorbers, while constraining both the total length of the network and the total area of the absorbers to be constant as functions of generation \( G \). A linearized form of the solution was put in dimensionless form which depends only on a dimensionless membrane resistance \( k \), a dimensionless inverse svelteness ratio \( \beta \), and \( G \). For each of the first nine generations \( G = 2, \ldots, 10 \), the optimal graph shapes were obtained. Total water production rate increases parabolically as a function of generation, with a maximum at \( G = 7 \). Total water production rate is shown to be approximately linearly related to the power consumed, for a fixed generation. Branching ratios which are optimal for desalination asymptote decreasingly to \( r = .510 \) for large \( G \), while branching angles which are optimal for desalination asymptote decreasingly to 1.17 radians. Asymmetric graphs were found to be less efficient for desalination than symmetric graphs. The geometry which is optimal for desalination doesn’t depend strongly on the dimensionless parameters, but the optimal water production does. The optimal generation was found to increase with the inverse svelteness ratio.

2.1 Introduction

Of the two most prevalent methods of desalination today, reverse osmosis and thermal distillation [64], reverse osmosis began relatively recently with the discovery in 1959 at UCLA by Loeb and others [39] of a chemically homogeneous,
physically asymmetric porous cellulose acetate polymer film [56] which made reverse osmosis economically feasible. Since typical energy requirements of reverse osmosis of $2.2kWh/m^3$ [65] are a factor of three larger than the theoretical limit of $0.76kWh/m^3$, other methods such as forward osmosis [22], low-temperature thermal desalination [9], and membrane distillation [60] continue to be introduced as viable alternatives. The byproducts of desalination include brine and mineral salts. Since these are acutely harmful to the environment, systematic studies of desalination from a Gibbs free energy perspective have become useful [69].

Current desalination research has continued to focus to a large extent on studying membranes. Promising results have been obtained from aquaporin [34] and carbon nanotube based [18] membranes, while analytical studies have yielded models for porous ion transport [17] and ion-exchange membranes [59]. Molecular dynamics studies dealing with ion layers in solution [77, 84] and osmosis through membranes [38, 54, 55] have also been popular. The optimality of fractal membranes [73, 71] has also been investigated, and in this connection it has been shown that membranes can be seen to be equivalent to electrodes [43]. Fractal antennae [33, 3] and battery electrodes [50] have also been shown to have optimal properties. It has been found that microscopic [36] and macroscopic [32] aggregates can spontaneously lead to ramified fractal networks, due to the optimality of the fractal configurations. Fractal growth networks which exhibit pattern formation under a reaction-diffusion dynamic have also been studied [75]. A useful mathematical description for the growth of fractal networks, iterated function systems [7], has been used to conveniently formalize and study visualization of fractals generated from chaotic sequences [70, 5].

The study and construction of ramified fractal-like networks and optimization of transport properties in the networks has been well investigated by many researchers. One of the foundational concepts which grounds such investigation is the principle that structures in nature and engineering adapt themselves to optimally serve their functions. In one of the seminal works illustrating this principle, Murray [48] showed how the cost of blood volume was the determining factor for the radii of a network of vessels transporting oxygen in man.
This has subsequently led to the development of a systemic study [8] of optimal flow configurations as a branch of non-equilibrium thermodynamics. Xu and others have recently shown how the thermal conductivity of both fractal tree-like branched networks [82] and of H-shaped fractal networks in composites [83], were significantly lower compared to conventional parallel channels. Similarly, Chen et.al. [16] have elucidated optimal branching diameter ratios which enhanced the effective permeability of composites over comparably sized parallel networks. Scaling laws for transport properties of conductive, convective, laminar, and turbulent flow fractal networks have also been presented [82]. In a related work, Mauroy and others [41] have determined the flow asymmetry versus branching angle for different aspect ratios and Reynolds numbers in a fractal-like tree network. We look at optimal desalination absorber networks. Our work is different than these earlier studies in that we have focused on the shapes themselves, unencumbered by the internal transport dynamics. In a sense we complement the previous work by helping impart a foundation for it. Furthermore previous work has not shown how optimal constructions arose from adaptation by absorbing networks to diffusive environments.

In the following, we explore the optimal geometry for networks of absorbers in a diffusive medium, applying this to the important case of desalination. We start by providing an explanation of what the system looks like in section II. In section III, we use the formalism of the iterated function system to generate a set of ramified graph networks of absorbers. The diffusion equation is solved by analogy to electrostatics in section IV, where we linearize the system and define the most relevant boundary condition to examine, that of constant pressure difference absorbers. In section V we discuss the results, as well as non-symmetric binary graphs, dependence of the solution on dimensionless parameters, and future research directions. Finally in section VI we summarize our results.

### 2.2 Physical Situation of the Posed Problem

Although we present a more formally detailed account of the technical details of the posed problem in what follows, it will be useful to initially give a down
Figure 2.1: Ramified graph which is optimal for (i.e. it maximizes) water production occurs for \( G = 7 \), \( r = .521 \), and \( \alpha = 1.16 \). The graph is optimal in the sense that other graphs with the same length and total area of absorbers, but with different branching angles and branching ratios, have lower water production rates.

to earth description of the physical situation. Our model is that of two planer, symmetric, binary trees, composed of a network of hollow pipes, joined at their base (see Fig. 2.1), embedded in a volume of saline solution. At the ends of the tree there are to be spherically shaped membrane surfaces (red outline in Fig. 2.2), which are permeable to water. Within the network of pipes near the base there are two symmetrically situated pumps which will thus produce a constant pressure difference at the absorbers. We assume that the absorbers are at the tips of the networks. It is known that the area of maximal absorption of plant roots occurs at their tips. However the biological shapes of roots are not spherical. Thus this would not be an entirely realistic assumption for a biological system. In order to better isolate the problem, we then dispense with the internal flow dynamics of the system, so that the diameters of the pipes play no role in what follows. Thus our principle consideration and result is to determine which branching angle, branching length ratio, and tree generation can most efficiently extract water from the pattern of salt concentration of the saline solution. Note that although the diffusion is in 3D, our patterns are confined to 2D. This would be more meaningful if we consider a system where the 2D desalination system is lying flat in a very shallow pool of sea water. In doing this we also make the further simplification that the saline solution
Figure 2.2: Schematic for a ramified network consisting of pipes with diameters \( w_{g,i} \), where the angles, ratios, and diameters depend on the generation \( g \). Red shading indicates effective membrane surface. For a network of total length \( l \), \( U_{g,i} \) and \( V_{g,i} \) denote \( h_{u_{g,i}} \) and \( h_{v_{g,i}} \) respectively.

has reached a steady state distribution of salt concentration. Since the rates of water production obtained (Fig. 2.5) are quite low, it is suggested that the assumption of steady state may be warranted.

### 2.3 Ramified Graphs and Iterated Function Systems

We use an iterated function system [7] which maps a set of \( 2^g \) line segments into a set with \( 2^{g+1} \) members:

\[
\phi : S_g = \{ A_{g,i} \} \rightarrow S_{g+1} = \{ A_{g+1,2i}, A_{g+1,2i-1} \}, \ g = 1, \ldots, G-1, \ i = 1, \ldots, 2^g
\]  

(2.1)

where \( A_{g,i} = \begin{pmatrix} u_{g,i} \\ v_{g,i} \end{pmatrix} \) represents the line segment \( \overline{u_{g,i}v_{g,i}} \) between nodes \( u_{g,i} \) and \( v_{g,i} \), and \( G \) is the total number of generations. The line segments of generation \( g \) are functions of the line segments of previous generations, that is:

\( A_{g+1,2i-1} = h_L(A_{g,i}) \) and \( A_{g+1,2i} = h_R(A_{g,i}) \), where \( h_L/R \) are linear transformations given in Appendix A, and \( A_{1,1} = (0, \frac{1-2r}{1-2r^2}, 0) \) is the stem segment.

The transformations \( h_{L/R} \) depend parametrically on the ratio \( r \) between segments of generation \( g+1 \) to segments of generation \( g \), and the angle \( \alpha \) with
which generation \( g + 1 \) segments branch away from generation \( g \) segments. Note that by the definition of \( A_{1,1} \) the total unitless length \( A_G = \sum_{g=1}^{G} \sum_{i=1}^{2^{g-1}} |A_{g,i}| \) is kept fixed at 1 as \( G \) is varied, i.e. \( A_G = 1 \) for all \( G \). The union of line segments forms a connected ramified graph \( G \): 

\[
G = \bigcup_{i \leq 2^{g-1}} A_{g,i}
\]

such that the starting point of the line segment of generation \( g + 1 \) is the endpoint of a line segment of generation \( g \), i.e. \( u_{g+1,2i} = u_{g+1,2i+1} \) = \( v_{g,i} \). The transformation \( h_{L/R} \) of Appendix A satisfies the relation 

\[
|u_{g+1,2i}v_{g+1,2i+1}| = |u_{g+1,2i}v_{g+1,2i+1}| = r \cdot |u_{g,i}v_{g,i}|
\]

therefore the iterated function is a contraction mapping for \( 0 \leq r < 1 \).

In what follows we assume that \( 0 \leq r < 1 \). The graph \( G \) models a ramified network of linear conduits, such as of pipes (see Fig. 2.2). If the total length of the network is \( l \), we may define \( U_{g,i} = l u_{g,i} \), and \( V_{g,i} = l v_{g,i} \). The endpoints \( S = \{V_{G,i}\} \) are the locations of absorbers, and \( U_{1,1} \) is an outlet for the permeate. The absorbers are semi-permeable membranes which use reverse osmosis or forward osmosis [15] to extract water from salt water.

### 2.4 Spatial dependence of the salt concentration

By conservation of salt molecules in a diffusive medium we have the equation

\[
\frac{\partial c}{\partial t} = -\nabla \cdot J.
\]

where \( c \) is the salt concentration in \( \text{mol/m}^3 \), and \( J \) is the flux of \( c \). Using Fick’s law,

\[
J = -D \nabla c,
\]

where \( D \) is the diffusion coefficient in units of \( \text{m}^2/\text{s} \). We thus find the diffusion equation for the system of water absorbers, 

\[
\frac{\partial c}{\partial t} = -\nabla \cdot (-D \nabla c).
\]

Assuming that \( D \) is constant, and that the system has reached a steady state, we get Laplace’s equation:

\[
D \nabla^2 c = 0.
\]

In solving Eq. (2.3), we assume the following boundary conditions: (i) the water surface is insulating, i.e., \( J(x = 0) = 0 \), and (ii) as the distance from the ramified graph approaches infinity the concentration goes to an ambient concentration \( c_\infty \). We furthermore assume that the end points \( \{v_{G,i}\} \) of the ramified
The centers of small spherical absorbers with radii $R_a$ which are sufficiently far apart that the salt concentration at the surface of each absorber is isotropic. Near an absorber $i$ the salt concentration is given by:

$$c(r) = \frac{s_i}{|r - \mathbf{v}_{G,i}|} + c_{i,x}(x - x_i) + c_{i,y}(y - y_i) + c_i^*, \quad (2.4)$$

where $s_i$ are integration constants with units of $\text{mol m}^{-2}$, $r = (x, y)$ is the location of measurement of the concentration $c$, $c_i^*$ is a constant, and $c_{i,x}$ and $c_{i,y}$ are first order derivatives which couple absorber $i$ to the other absorbers in the network.

Using Gauss’ law and integrating over a spherical region $\Omega$ of radius $r$ centered at a single salt absorber of radius $r > R_a$, the production rate $Q_i$ of salt in units mol s$^{-1}$ coming from a single absorber $i$ is given by:

$$Q_i = \int_{\partial\Omega} \mathbf{J} \cdot \hat{n} dA = 4\pi D s_i \quad (2.5)$$

where $J$ is computed by applying Eq. (2.2) to Eq. (2.4). Gauss’ law shows that the constant $s_i$ is proportional to the salt production rate $Q_i$ coming from a single absorber. So since (2.3) is linear, the principle of superposition gives an approximate complete solution to Eq. (2.3):

$$c_i = \frac{2^{G/2}Q_i}{D\sqrt{4\pi A_a}} + \frac{1}{4\pi D l_j \sum_{j \neq i} \mathbf{v}_{G,j} - \mathbf{v}_{G,i}} Q_j + c_\infty. \quad (2.6)$$

where $c_i$ is defined as the salt concentration at the $i$th absorber, $i = 1, \ldots, 2^G$, and $A_a = 2^G 4\pi R_a^2$ is the total surface area of the absorbers.

The salt concentration $c_i$ at the outside of the membrane of absorber $i$ depends on the applied pressure drop $\Delta_i$ at each absorber membrane. In what follows, we consider the case where $\Delta_i = \Delta$ is fixed for all $i$, which is physically the most easily realized case. The pressure difference across the membrane $\Delta$ is the sum of the osmotic pressure and the membrane flow resistance $\Delta_m$,

$$\Delta = (c_i - c_0)RT + \Delta_m = c_i^*RT + \Delta_m, \quad (2.7)$$
where \( c_0 \) is the salt concentration inside the absorber, \( c'_i = c_i - c_0 \) is the difference in concentration from outside to inside the absorber, \( \Delta_m = \frac{W_i \mu b}{4\pi \kappa R^2} \) is the membrane flow resistance, the water production rate \( W_i \) is the volume of water flow through absorber \( i \) per unit time, \( \mu \) is the viscosity of the medium in kg m\(^{-1}\)s\(^{-1}\), \( \kappa \) is the membrane permeability in m\(^2\), \( b \) is the thickness of the membrane, \( R = 8.314 \text{ J/K mol} \) is the ideal gas constant, and \( T \) is the temperature in Kelvin. By the stoichiometry of the molecules which interact with the absorbers, the rate of water production \( W_i \) at node \( i \) is given by the relation

\[
Q_i = W_i c'_i
\]  

(2.8)

where \( Q_i \) is the salt production rate of node \( i \). Hence we have that the total rate of water production \( W \) for the absorber network is just:

\[
W = \sum_{i=1}^{2^G} Q_i
\]  

(2.9)

Solving Eq. (2.7) for \( W_i \), the power consumption \( P_i = \Delta W_i \) at absorber \( i \) is just

\[
P_i = (c'_i RT + \Delta_m)W_i,
\]  

(2.10)

and the energy consumption \( E_i = P_i/W_i \) is given by:

\[
E_i = c'_i RT + \Delta_m
\]  

\[
= \Delta, \quad i = 1, \ldots, 2^G.
\]  

(2.11)

Using Eq. (2.7) and Eq. (2.8), the salt production rate \( Q_i \) is given by an expression quadratic in \( c'_i \):

\[
Q_i = \frac{\kappa A_a}{\mu b 2^G} (c'_i \Delta - c'_i^2 RT).
\]  

(2.12)

Substituting these expressions for \( Q_i \) into Eq. (2.6) then gives a quadratic form in the \( c'_i \):

\[
\frac{\mu b D}{\kappa} (c_i - c'_i) + \sqrt{\frac{A_a}{2^G 4\pi}} (c'_i \Delta - c'_i^2 RT) + \frac{A_a}{2^G 4\pi} \sum_{j \neq i} e_j \Delta - e_j^2 RT = 0,
\]  

(2.13)
where \( c_\delta = c_\infty - c_0 \) is the difference in salt concentration between the inside of the absorber and infinity. Eq. (2.13) constitutes our steady state solution of the diffusion equation, for the boundary case of constant pressure difference at the absorbers and a constant ambient salt concentration far from the absorbers. Note that our procedure for solution, which follows the analogy of diffusion to electrostatics originated by Maxwell [42] is in essence an elementary Green’s function method for the solution of Laplace’s equation.

Defining a dimensionless concentration

\[
\tilde{c}_i = \frac{c'_i - \frac{\Delta}{RT}}{c_\delta - \frac{\Delta}{RT}},
\]

(2.14)

Eq. (2.13) may be rewritten in dimensionless form as:

\[
\tilde{c}_i + \frac{k}{\sqrt{4\pi 2^d}} \tilde{c}_i (1 - \xi \tilde{c}_i) + \frac{k\beta}{4\pi 2^d} \sum_{j \neq i} \sum_{G,j} \left| \mathbf{v}_{G,j} - \mathbf{v}_{G,i} \right| = 1,
\]

(2.15)

where the dimensionless parameters of the problem are

\[
k = \sqrt{\frac{A_a \kappa \Delta}{\mu b D}},
\]

(2.16)

\[
\beta = \sqrt{\frac{A_a}{l}},
\]

(2.17)

\[
\xi = \frac{\Delta - c_\delta RT}{\Delta},
\]

(2.18)

and \( G \). We observe that each of these dimensionless parameters has a physical interpretation. \( k \) is an effective dimensionless membrane resistance, which relates the applied pressure induced resistive flow across the membrane to the diffusivity \( D \) of the medium, \( \beta \) is an inverse svelteness ratio which gives how large the fixed total absorber area \( A_a \) is in relation to the fixed total length \( l \), and \( \xi \) is the normalized applied pressure. From Eq. (2.8) and Eq. (2.12), we can then write the rate of water production \( W_i \) for node \( i \) as:

\[
W_i = \frac{1Dk\beta\xi}{2\delta} \tilde{c}_i.
\]

(2.19)

For typical desalination values given in Table 2.1, on average \( \xi \tilde{c}_j < .03 \) for
<table>
<thead>
<tr>
<th>Desalination Parameter</th>
<th>Description</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Diffusion constant</td>
<td>$10^{-9}$ m$^2$/s</td>
</tr>
<tr>
<td>$T$</td>
<td>Temperature</td>
<td>290 K</td>
</tr>
<tr>
<td>$c_\delta$</td>
<td>Background salt concentration</td>
<td>564 mol/m$^3$</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>Applied pressure difference</td>
<td>$1.4 \times 10^6$ Pa</td>
</tr>
<tr>
<td>$\frac{\delta}{\mu}$</td>
<td>Effective membrane permeability</td>
<td>$9.72 \times 10^{-20}$ m$^3$/s/kg</td>
</tr>
<tr>
<td>$A_\alpha$</td>
<td>Total membrane area</td>
<td>.00025 m$^2$</td>
</tr>
<tr>
<td>$b$</td>
<td>Total membrane thickness</td>
<td>$10^{-6}$ m</td>
</tr>
<tr>
<td>$l$</td>
<td>Total network length</td>
<td>1.0 m</td>
</tr>
</tbody>
</table>

Table 2.1: Typical desalination values used to solve Eq. (2.3).

generations up to $G = 10$. Therefore Eq. (2.15) can be linearized with a Taylor series expansion about $\hat{c}_i = 0$ (see Appendix B):

$$(1 + \frac{k}{\sqrt{4\pi 2^d}})\hat{c}_i + \frac{k\beta}{4\pi 2^d} \sum_{j \neq i} \frac{\hat{c}_j}{|v_{G,j} - v_{G,i}|} = 1. \quad (2.20)$$

In order to maximize the total water production rate $W$, given the fixed applied pressure $\Delta$, the system given by Eq. (2.20) must be solved. Levy’s theorem [74] suggests that a solution exists (see Appendix C) except when absorbers overlap. Fig 2.1 shows the ramified graph corresponding to the ramified network which is optimal for water production rate since for all angles and ratios considered it has the largest water production rate. In all cases for the values given in Table 2.1, the error made by neglecting the second order term (B.6), which we take as the ratio of the Euclidean norm of the second order term vector to the Euclidean norm of the first order term vector (see Eq. (B.5)), was less than $6 \times 10^{-3}$.

### 2.5 Discussion

In our study we seek to explore morphologies which produce maximal water production rates. Since a desalination system is considered most desirable if it maximizes water production (ceteris paribus), we thus term such graphs possessing this quality ”optimal.” The optimal ratios and angles respectively as functions of generation corresponding to the optimal ramified graphs are given
Figure 2.3: Ratio for ramified graph networks having maximal water output for the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. Ratios were deemed optimal if the corresponding water production rate was maximal over all angles.

In Figs 2.3-2.4. By varying angle and ratio independently and calculating the water production for each generation, we were able to thereby find the optimal ratios and angles in the sense that they maximized water production rates. By fitting the optimal ratio and angle plots, we were able to evaluate their asymptotic values. In Fig 2.3 it is demonstrated that the optimal ratio decays asymptotically to the value of .510, which as an approximation to .5 is the value one might expect based upon symmetry considerations. Fig 2.4 shows that the optimal branching angles asymptote exponentially to the value of 1.17, which is about 67 degrees. In Fig 2.5 we give the optimal water production rates as a function of the generation. The criterion for optimization was that the graph produced the maximum volume of water per unit time. Hence Fig 2.5 plots the actual maxima that were found by varying ratio and angle as generation $G$ was varied. Fig 2.5 shows that the optimal water production increases up to a maximum at about $G = 7.1$. In Fig 2.6 are given all the optimal graphs for generations $G = 2$ to $G = 10$. For clarity, we superimpose the graphs in groups of three. Plotting graphs three generations apart on the same plot (e.g. generations 2, 5, and 8) enables one to distinguish the features of each individual graph better (as opposed to e.g. plotting generations 2,3, and 4 together). For each
Figure 2.4: Angle for ramified graph networks having maximal water output for the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. Angles were deemed optimal if the corresponding water production rate was maximal over all ratios.

Figure 2.5: Optimal water production rates in m$^3$/s for ramified graph networks in the case of constant pressure difference absorbers as function of number of generations, for generations $G = 2$ to $G = 10$, with parameter values as in Table 2.1. These are the values of water production per unit time that were maxima (and hence optimal) for each generation by independently varying ratios and angles.
Figure 2.6: Ramified graphs having optimal properties (maximal water production rate for constant pressure difference absorbers) as in Figs. 2.3-2.4 for generations $G = 2$ to $G = 10$. The graphs are optimal in the sense that other graphs with the same length and total area of absorbers, but with different branching angles and branching ratios, have lower water production rates.
separate graph, we apply the iterated function system with inputs the specified generation $G$, and the optimal ratio $r$ and angle $\alpha$ for that generation. The lengths of all these nine graphs are the same. The optimal ratios and angles are those values which yield the maximum water production for the corresponding desalination system.

Fig 2.7 show contours of the total water production rate of the desalination system versus angle and ratio for generations 3, 5, and 7. It is noteworthy from the contour plots of water production, Fig 2.7, that as generation $G = 7$ is reached, the possible solutions to the system becomes increasingly spatially restricted (the maximum is sharp), so that by generation $G = 10$ the system is only solvable (see Appendix C) for a low percentage of cases. It is evident from the contour plots that the optimal water production which occurs at about generation $G = 7.1$ is a type of local maximum. Another intriguing aspect of these plots is that the lines of singularity correspond to cases where the matrix was not invertible. Taking cross-sections through the maximum by holding either the branching angle or branching ratio constant at their optimal values it is possible to determine how the water production changes as a function of angle or ratio. This behavior is found to be parabolic in both cases as depicted in Fig 2.8, which also enables us to determine at which angle and length ratio that the water production rate can reach the maximum value. In Fig 2.9 the relation between the optimal total water production rate $W$ and the energy consumption of the desalination nodes, given by Eq. (2.11), is shown to be a nearly linear, increasing function.

Previous work has shown that termini of ramified transportation networks are binary [32], so we have restricted our investigation to binary graphs. It may be suggested that only a special case of binary graphs has been examined, and that the class of binary graphs may have other more optimal geometries than our solutions. In the following however, we give an indication that the symmetric binary graphs generated by the iterated function system of Eq. (A.1) yield the optimal geometries amongst more general binary graphs, as is suggested by the most symmetric asymptotic optimal ratios and angles of the results. Indeed, we varied the difference in branching ratio in the range $-1 \leq r_L - r_R \leq 1$
Figure 2.7: Contours of total water production for constant pressure difference absorbers over the ramified graph network for equal increments of angles between 0 and $\pi$, and ratios of 0 to .9, for generations $G = 3, 5, 7$, with parameter values as in Table 2.1. Lines of singularity occur where the coefficient matrices were non-invertible.
Figure 2.8: Plots of water production $W$ versus ratio $r$ with $G = 7$ and $\alpha = 1.167$ (top), and water production $W$ versus angle $\alpha$ with $G = 7$ and $r = .5155$ (bottom).
Figure 2.9: Optimal water production rate in m$^3$/s for $G = 7$ as a function of the energy consumption per volume $E$ of permeate produced.

while keeping the branching angle fixed at the optimal angle for $G = 3$, and found that the optimal water production occurred when $r_L - r_R = 0$, which is the case of our symmetric binary graph (Fig. 2.10 top). We similarly studied how changing the branching angle in the range of $-2.19 \leq \alpha_L - \alpha_R \leq 2.19$, with branching ratio kept at the optimal value for $G = 3$, and found that the symmetric binary graphs $\alpha_L - \alpha_R = 0$ produced the greatest water production (Fig. 2.10 bottom). Thus our results also suggest (although this verification is by no means exhaustive) that the symmetric types of binary graphs which Eq. (A.1) generate have the most optimal properties.

Another modification in the assumptions which conceivably might change our results is to alter one or more of the parameters of the problem. In order to eliminate any dependence on scale, only dimensionless parameters of the problem should be changed. Referring to the linear system of Eq. (2.20), we have that the two dimensionless parameters are $\beta$ and $k$, given by Eqs. (2.17) and (2.16) respectively. Evaluating Eq. (2.17) and Eq. (2.16) with the typical values given in Table 2.1 give

$$\beta = .0158,$$

and

$$k = 2.15.$$
Figure 2.10: Optimal water production for $G = 3$ versus ratio difference (top) and angle difference (bottom). For the ratio difference plot, alpha was the optimal angle, and ratios were varied independently about the optimal ratio. For the angle difference plot, ratio was the optimal ratio, and angles were varied independently about the optimal angle.
If we thus define $\beta_0 = 0.0158$ and $k_0 = 2.15$, we see that $\beta_0$ and $k_0$ signify the typical values of the dimensionless parameters $\beta$ and $k$. Defining $\beta_d = d\beta_0$ and $k_d = dk_0$, we then studied the solutions to the linear system of Eq. (2.20) by alternately holding $k = k_0$ constant with $\beta = \beta_d$ varying, and then $\beta = \beta_0$ constant while varying $k = k_d$. In each case $d$ was varied between 0.5 and 1.5, in steps of 0.25. Perhaps not so surprisingly in light of our systematic study of dependence of optimality on geometry above, the optimal ratios and angles did not change for these different values of $(\beta, k)$. However, the optimal water production rates did change. In order to see this we produced curve fits for the optimal water production plots (e.g. for plots like that of Fig. 2.5) by the function

$$W(G) = C_1 - C_2(G - G_{\text{max}})^2.$$  \hspace{1cm} (2.21)

For all of the different parameter combinations of $(\beta, k)$ the maximal water output was found to have an inverted parabolic dependence on the generation $G$, as shown by the curves in Fig. 2.11. In fact, the value of $G_{\text{max}}$ gives by extrapolation a value for the optimal (in the sense of maximal water production) generation. Thus although the parameters $\beta$ and $k$ don’t appear explicitly in Eq. (2.21), it is shown in Fig. 2.12 how the optimal generation $G_{\text{max}}$ of Eq. (2.21) changes as the parameters $\beta$ and $k$ are varied. Indeed, Fig. 2.12 demonstrates how $G_{\text{max}}$ has an increasing (implicit) dependence on $k$, while $G_{\text{max}}$ decreases with increasing $\beta$. This shows that smaller absorbers favor a higher generation.

### 2.6 Conclusion

Beginning with an iterated function system which was used to generate the ramified graph networks of absorbers (Fig 2.2), the diffusion equation was solved using the analogy to electrostatics. After introducing dimensionless parameters which had direct physical correlates, the system was linearized for the case of constant pressure difference absorbers, experimentally the most accessible case. The linear system of $2^G$ equations was then solved for the $2^G$ unknowns, specifying the concentrations about each of the $2^G$ absorbers in the network, for generations $G = 2$ to $G = 10$ while independently varying branching ratios.
Figure 2.11: Curve fits of optimal water production while varying independent dimensionless parameters. On the top is shown the effect of varying $\beta$, while the effect of varying $k$ is shown at bottom. All curves were fit to the general parabolic form of $W(G) = C_1 - C_2(G - G_{max})^2$. 

\[ W \text{ in } \text{m}^3/\text{s} \]

\[ k = 0.5k_0 \]
\[ k = 0.75k_0 \]
\[ k = k_0 \]
\[ k = 1.25k_0 \]
\[ k = 1.5k_0 \]
Figure 2.12: Dependence of $G_{max}$ from Eq. (2.21) on the dimensionless parameters $\beta$ (top) and $k$ (bottom).
and angles. The optimal ramified graph, which resulted in the greatest water production rate all other quantities being equal, is shown in Fig 2.1. Optimal ratios, angles, and water production rates were obtained for each generation \( G \). Contour plots showed how the solutions smoothly approached the same optimum as \( G \) was increased, thus demonstrating that information was not lost as the system grew in size and became less soluble (see Appendix C). By independently varying the left and right branching angles and ratios, it was shown why our symmetric ramified graphs were preferable to the asymmetric variants from the point of view of optimization. Changing the values of the independent dimensionless parameters demonstrated that the geometry of the optimal graphs did not change, but that the optimal generation decreased as the scaling factor increased. The findings we present here also show that the smaller the absorber area, the higher the optimal generation. The foregoing may help design more efficient networks and provide optimal shapes for practical desalination systems to increase the supply of drinking water in the world.
Chapter 3

Further Investigation of Optimal Desalination Graphs

3.1 Other Boundary Conditions for Model of Chapter 2

Beyond the boundary case of constant pressure difference absorbers treated in Chapter 2, we here discuss other boundary conditions for the same model and their solution.

3.1.1 Constant Salinity Absorbers

For the boundary case of constant salinity absorbers we assume that the salinity $s_i$ are all constant, that is that $s_i \equiv s$, $i = 1, \ldots, 2G-1$. This in turn implies by Eq. 2.5 that the rate of salt production is constant for all nodes:

$$ Q_i \equiv Q = 4\pi D s, \; i = 1, \ldots $$

(3.1)

Since $s_i = s$ is constant, the salt concentration for node $i$ by Eq. 2.6 is just:

$$ c_i = s\left(\frac{1}{R_a} + \sum_{j \neq i} \frac{1}{|v_{G,j} - v_{G,i}|}\right). $$

(3.2)

Now Eqs. 2.8, 2.5, and 3.2 give the rate of water production of node $i$:

$$ W_i = \frac{4\pi D s}{s\left(\frac{1}{R_a} + \sum_{j \neq i} \frac{1}{|v_{G,j} - v_{G,i}|}\right)} $n

$$ = \frac{4\pi D}{\frac{1}{R_a} + \sum_{j \neq i} \frac{1}{|v_{G,j} - v_{G,i}|}} $$

(3.3)
Thus in this case, the total rate of water production, \( W_{\text{tot}} \), is given by:

\[
W_{\text{tot}} = \sum_i W_i = 4\pi D \sum \frac{1}{R_a} + \sum_{j \neq i} \frac{1}{|\mathbf{v}_{G,j} - \mathbf{v}_{G,i}|} \tag{3.4}
\]

We then maximize \( W_{\text{tot}} \), given a fixed salinity \( s_i = s, \ i = 1, \ldots \)

**Optimal Absorber System**

Note that in the solutions for the different boundary cases given above, it was generally the case that analytical solutions would have been fairly involved to compute. However for the last case, that of constant salinity absorbers, it is quite straightforward to solve the equations.

Thus we investigate the possibility of there being extrema at generation 2 for the case of constant salinity absorbers. From Eq. A.1, we have, with \( \mathbf{v}_0 = (0, d_1) \),

\[
\mathbf{v}_{0L} = (d_1 \cos \left(\frac{\pi}{2} + \alpha\right), d_1(1 + r \sin \left(\frac{\pi}{2} + \alpha\right)))
\]

\[
= (-d_1 \sin \alpha, \ d_1(1 + r \cos \alpha)), \tag{3.5}
\]

and

\[
\mathbf{v}_{0R} = (d_1 \cos \left(\frac{\pi}{2} - \alpha\right), d_1(1 + r \sin \left(\frac{\pi}{2} - \alpha\right)))
\]

\[
= (d_1 \sin \alpha, \ d_1(1 + r \cos \alpha)). \tag{3.6}
\]

Applying Eq. 3.3, we have

\[
W_1 = \begin{cases} 
4\pi D \frac{1}{R_a} + \frac{d_1 r \sin \alpha}{2|d_1 r | \sin \alpha |} & \text{if } d_1 r |\sin \alpha| > R_a, \\
2\pi D \frac{1}{R_a} & \text{if } d_1 r |\sin \alpha| \leq R_a,
\end{cases} \tag{3.7}
\]

so that symmetry and Eq. 3.4 give:

\[
W_{\text{tot}} = 2W_1. \tag{3.8}
\]
To find the maximum water production of the network of absorbers, we then differentiate $W_{tot}$ with respect to the parameters $r$ and $\alpha$, finding:

$$\frac{\partial W_{tot}}{\partial r} = -2\left(\frac{4\pi D}{R_a + \frac{1}{2d_1r|\sin\alpha|}}\right)^2\left(\frac{1}{2d_1r|\sin\alpha|}\right),$$  

(3.9)

and

$$\frac{\partial W_{tot}}{\partial \alpha} = -2\left(\frac{4\pi D}{R_a + \frac{1}{2d_1r|\sin\alpha|}}\right)^2\left(\frac{1}{2d_1r|\sin\alpha|}\right) \cos\alpha.$$  

(3.10)

For fixed angle $\alpha$, Eq. 3.9 has no zero, thus $W_{tot}$ has no local maxima as a function of $r$, although there is an absolute maximum when $r = r_{max} = .9$. On the other hand, $W_{tot}$ is seen to have a local maximum as a function of $\alpha$ with $r$ fixed at $\alpha = \pi/2$. Both of these results are in agreement with simulation.

**Numerical Optimization of Absorber System**

We present here the results of simulation to solve for constant salinity absorbers. In Fig 3.1 is shown the total water production rate of the desalination system versus angle and generation. In Fig 3.2 we give the optimal angles and ratios as functions of angle for the case of constant salinity absorbers. Fig 3.3 shows the ramified graphs corresponding to the optimal ramified networks of Fig 3.2.

**3.1.2 Constant Water Product Absorbers**

In the case of constant water product absorbers we assume all nodes produce water at the same rate $W$, i.e. $W_i = W$, $i = 1, \ldots$ is fixed. So the total water production given $n$ nodes is just

$$W_{tot} = nW.$$  

(3.11)

Thus, from Eqs. 2.8, 2.5, and 2.6,

$$c_i = \frac{1}{W}4\pi Ds_i = \frac{s_i}{R_a} + \sum_{j\neq i} \frac{s_j}{|v_{G,j} - v_{G,i}|}.$$  

(3.12)
Figure 3.1: Total water production for constant salinity absorbers over the ramified graph network for equal increments of angles between 0 and $\pi$, and ratios of 0 to .9, for generations $g = 2$ to $g = 10$. Note that since the boundary conditions are different in this case as for those for Fig. 2.7, the fine structures (lines of singularities) of the plots are largely non-existent.

Figure 3.2: Angle (blue/dashed) and ratio (red/solid) for ramified graph networks having maximal water output for the case of constant salinity absorbers as function of number of generations, for generations $g = 2$ to $g = 10$. 
Figure 3.3: Ramified graphs having optimal properties (maximal water production rate for constant salinity absorbers) for generations $g = 2$ to $g = 10$.

Rearranging, we then get the linear homogeneous system in $s_i$, $i = 1, \ldots$

$$
\left( \frac{1}{R_a} - \frac{4\pi D}{W} \right) s_i + \sum_{j \neq i} \frac{s_j}{\|v_{G,j} - v_{G,i}\|} = 0 \quad (3.13)
$$

Given $W_i \equiv W$, we solve Eq. 3.13 for $s_i$, $i = 1, \ldots, 2^{G-1}$ and then minimize the mean pressure, which by Eq. 2.11 thus optimizes the energy consumption of the system.

The method of solving Eq. 2.3 for constant pressure difference absorbers is similar to that of the case of constant salinity absorbers, with the difference that now it is necessary to establish invertibility of the matrix of coefficients. The resulting matrix is by definition real, positive, and bi-symmetric, with the main diagonal being an element larger than the other elements. Theoretically, it would be desirable to find conditions on the matrix such that it would be invertible. We find with simulations that in some cases the matrix is not invertible for some values of the ratios and angles, but that in most "typical" cases the matrix is invertible. If the matrix is not invertible, then there will not be one unique solution to the salinity, but an infinitude of them—this would present additional issues that we seek to avoid for the sake of simplicity.

For the case of constant water product absorbers, that of constant permeate (water) production rate, Eq. 3.13, being homogeneous, will have either zero solutions or an infinite number of solutions, if we omit the trivial solution which would be useless for our purposes. Thus we expect an infinite number of solutions in this case, and therefore need an additional constraint to obtain a
3.2 Further Investigations

In the following we discuss other related work which may follow the project treated in Chapter 2 and above.

3.2.1 Optimal Graphs by Optimizing All Angles, Ratios, and Weights

Define $\alpha_g$ as the absolute value of the angle between $A_{\sigma_g}$ and $A_{\sigma_g L/R}$ (see Fig. 3.4). Similarly, define $r_g$ to be the radius of pipe segment $A_{\sigma_g}$. We intend to further extend our results by making the angles $\alpha_g$ and ratios $r_g$ functions of the generation. Taking this one step further, we could also optimize $\alpha_{g,i}$ ($r_{g,i}$), the absolute value of the angle between $A_{\sigma_{g,i}}$ and $A_{\sigma_{g,i} L/R}$, (radius of $A_{\sigma_{g,i}}$), for $i = 1, \ldots, 2^g$ the identity of node $i$. Furthermore, we here introduce the notion of a weight $w_{R/L}(\alpha_g, r_g)$ for the line segments of generation $g$. Interpreted corresponding to the piping of a desalination system, the weight could be a function which gives the diameter of the pipe as a function of the angle $\alpha_g$ and ratio $r_g$ (see Fig. 3.4). In this case we have for the length which we fix:

$$L = \sum_{g=0}^{G-1} |u_0 v_0| 2^g r_g^g.$$  \hspace{1cm} (3.14)
Alternatively, using weights we may opt for a fixed total mass of the graph:

\[ M = \sum_{\sigma} w_\sigma |\mathbf{u}_\sigma \cdot \mathbf{v}_\sigma| \equiv \text{constant}. \tag{3.15} \]

In this case (where we use weights), for optimization of power consumption there will be additional terms in the equation for total power, given for viscous incompressible flow of a fluid in pipe according to the Hagen-Poiseuille equation by:

\[ P_\sigma = \frac{|\Delta P_\sigma|^2 \pi w_\sigma^4}{8\eta |\mathbf{u}_\sigma \cdot \mathbf{v}_\sigma|}, \tag{3.16} \]

where \( \Delta P_\sigma \) is the pressure drop along \( |\mathbf{u}_\sigma \cdot \mathbf{v}_\sigma| \), and \( \eta \) is the dynamic fluid viscosity in units of \( Pa \cdot s \). Interesting questions which arise in this context include: does the optimal \( \alpha_g \) relate to the golden mean? or does the optimal angle equal a constant, i.e. does \( \alpha_g \equiv \alpha \)?

For this we may need to use more powerful numerical optimization techniques, such as gradient search or discrete simulated annealing. Using the same search methods as we are currently employing, the number of generations will have to be limited in order to obtain results in a reasonable computational time.

### 3.2.2 Self-Adjusting Graphs

Another extension of our results will be to use a neural network learning algorithm to obtain fully optimized parameters for all properties of a ramified graph. Again we will use as our example the case of a system for desalination. For this we will define a notion of resistance, and a new measure for size, such as the diameter of the pipe. In this way the pipe diameter can act as a neural network weight, which will be altered in some definable way in order to obtain optimal properties for the three boundary conditions. We will fix the total mass of the pipe as given in Eq. 3.15. The algorithm will find ramified graphs which are optimal by changing the weights and positions of the absorbers. In addition, we could introduce the notion of heterogeneous absorbers. This brings up the possibility of determining whether, given an imposed salt concentration gradient, does the graph adapt so that branches towards lower concentration regions are preferred, thus breaking the symmetry of the graph?
3.2.3 3-D Graphs

By adding a second angle, it is possible to specify a three-dimensional non-intersecting ramified graph. We will extend our results for optimal properties of the two dimensional desalinating ramified network to that of a three dimensional network. For this case absorbers may either be at the endpoints of the graphs or we may specify that the branches of the graph act as absorbers. In this connection it is useful to note that the shape of a particular marine plant, the red mangrove (Rhizophora Mangle), see Fig. 3.5, has a three dimensional fractal-like shape similar to our two-dimensional graphs. Although most plants which perform desalination accomplish this by active transport, Rhizophora Mangle is thought to achieve reverse osmosis through its roots [61]. This is not a coincidence since the origin of Lindenmayer systems, which have similar properties to our ramified graph, is in modelling the vascular systems of plants. Thus, since it is known that the spatial branching angles of plant root systems are related to the golden mean, an interesting possibility is that we would be able to show that the optimal branching angles of the ramified graph will be similarly related to the golden mean.
3.2.4 Experiments on Desalination

Two different experiments are proposed, one experiment with electrolysis and another using reverse osmosis.

Desalination by Reverse Osmosis

One straightforward experimental approach would be to build the desalination system using pipes and polymer absorbers. The goal of the experiment would be to validate the results obtained by simulation and theoretically. When investigating the boundary condition that the water production rates are all equal, there will need to be a small pump placed inside of each absorber, and the pressure produced homeostatically controlled in order to maintain the constancy of a water flow-rate indicator. An easier boundary condition to examine in the lab would be the case of constant pressure at the absorbers, since in this case only one large pump would be placed at the sink, \( u_0 \). Most difficult of all, however, would be to evaluate the performance of the optimal design under the conditions of constant salinity of the absorbers, since in this case the salinity level at the outside surface of the absorber will have to be monitored, and feedback controls to the pressure pumps would be utilized to dynamically modify the pressure level while maintaining the salinity constant. In each case agreement between experiment and theory will be compared over a range of average salt concentrations.

Desalination by Electrolysis

The other proposed experiment will be using electrolysis to desalinate a saline solution. For this it will be necessary to establish how the two problems are in some sense thermodynamically equivalent. I.e., we will need to show that an optimal ramified network of absorbers performing reverse osmosis will, given the same spatially optimized shape, also be optimal for electrolysis of water, and vice versa. Then we will be able to validate the optimization results with electrode structures of ramified networks using electrolysis.

The experiment uses a concentration cell consisting of two containers of electrolyte (NaCl in water), one of higher concentration than the other, each
with a standard electrode placed in it, the negative in the high concentration electrolyte, the positive in the low concentration electrolyte, and connected by a salt bridge (see Fig. 3.6). After the hydrogen gas from the reduction is collected, it will be oxidized to yield the purified water. In this process we will harvest the heat of combustion to make the desalination more efficient. It is hoped that thereby we may obtain an efficiency which is comparable to typical values for reverse osmosis, e.g. a few kilowatt-hour per m$^3$ of water purified.
Chapter 4

Learning rate and attractor size of the single-layer perceptron.


4.1 Abstract

We study the simplest possible order one single-layer perceptron with two inputs, using the delta rule with online learning, in order to derive closed form expressions for the mean convergence rates. We investigate the rate of convergence in weight space of the weight vectors corresponding to each of the 14 out of 16 linearly separable rules. These vectors follow zigzagging lines through the piecewise constant vector field to their respective attractors. Based on our studies, we conclude that a single-layer perceptron with \( N \) inputs will converge in an average number of steps given by an \( N \)th order polynomial in \( t \), where \( t \) is the threshold, and \( l \) is the size of the initial weight distribution. Exact values for these averages are provided for the five linearly separable classes with \( N = 2 \). We also demonstrate that the learning rate is determined by the attractor size, and that the attractors of a single-layer perceptron with \( N \) inputs partition \( \mathbb{R}^N \oplus \mathbb{R}^N \).

4.2 Introduction

Since its advent in 1960, the study of the perceptron and of neural networks has experienced vast growth. Some fundamental characteristics about perceptrons were studied initially, while more recently several sophisticated and subtle questions have been addressed by the physics community. One significant current
of research is in online learning, where amongst other aspects that of ensemble learning (combination of many different learning rules) of simple perceptrons has been examined [45]. More recently there has been an interest in studying the effects of noise on learning in recurrent perceptron networks [47], and aspects of convergence for perceptrons with stochastic, binary synapses [63].

One of the tools used to analyze perceptron generalization and capacity is the replica method [79], and this technique has been used to determine the optimal capacity of ternary perceptrons [12], the generalization and capacity of large two-layered perceptrons [57], as well as to study learning capability under mutual information maximization [76].

Many other techniques for studying neural networks have appeared in the recent physics literature. For instance, one important area of current physics research treats the issue of generating and learning time series, including chaotic time series, by perceptrons [24, 53, 52, 62]. The study of time series generation by the perceptron is notable in addition as they have been used to produce limit cycles [62]. Information theory approaches have also been fruitful, in yielding storage capacity values for neural networks with binary weights in good agreement with the replica method [72], as well as an information gain principle which can give insight on how to choose training sets and transfer functions for student-teacher learning paradigm perceptrons [19]. Moreover, it is remarkable that perceptron learning has also being applied to biophysics problems, in particular in the context of improving the pairwise contact energy function in the study of the protein folding problem [27].

There are numerous convergence theorems which demonstrate that a perceptron learning a linearly separable rule will converge in a finite number of steps [80]. Recently, analogous upper bounds for the number of steps have been proven for continuous perceptrons using online gradient learning [81], and boolean multilinear perceptrons [14]. In addition, systematic studies have been made of the rate of convergence for perceptrons learning non-separable rules [10], and order of magnitude comparisons were given for learning rates for batch, online, and cyclic learning [28].

There have been, however, no studies of visualizing the convergent weight
dynamics of perceptrons, and very few in studying quantitatively the number of steps required to converge within weight space. In this paper we address these issues by giving concrete results from direct calculations, which quantify precisely the mean number of steps the weight vector travels in weight space according to which (linearly separable) rule it is learning. Our paper is organized as follows. In the next section, we discuss in detail the exact algorithm we used to obtain our results, and explain how the perceptron learning procedure itself necessarily fills weight space completely with its attractors. In the section on convergence rates, we outline our calculation for the average convergence rates, and state these for \( N = 2 \). Finally we discuss some implications of these results, and in the conclusion offer some possible applications and directions for future research.

### 4.3 Perceptron Algorithm

We use a cyclic online binary perceptron algorithm [28], which we outline as follows. A binary perceptron implements a boolean function which maps \( \mathbb{Z}_2^N \) to \( \mathbb{Z}_2 \). The input vectors are represented by the binary numbers \( x^j = x_1 2^{N-1} + x_2 2^{N-2} + \ldots + x_N \), where \( x_i \in \{0, 1\} \), and \( 0 \leq j \leq 2^N - 1 \). There are \( 2^N \) possible input patterns (vectors) since each component can either be 0 or 1. The weight vectors are written as \( w_n^k \in \mathbb{R}^N \), \( 0 \leq n < \infty \), where \( k \) corresponds to a rule, to be defined below. Here \( n \) is to be thought of as the discrete time, and there are \( 2^N \) weight components since each input (a component of the input vector) is weighted by a corresponding component of the weight vector. The desired output for each of the \( 2^N \) input vectors \( x^j \) is denoted by \( d_j \in \mathbb{Z}_2 \), \( 0 \leq j \leq 2^N - 1 \).

The actual output of the neuron \( y \) is given by:

\[
y(w_n^k, x^j) = \Theta(w_n^k \cdot x^j + (-1)^{\lceil 2^N-1 \rceil t}), \quad 0 \leq k \leq 2^N - 1
\]

Here \( (-1)^{\lceil 2^N-1 \rceil t} \) is the threshold (\( \lceil z \rceil \) denotes the smallest integer greater than or equal to \( z \)) and \( \Theta(x) \) is the Heaviside function, so that the perceptron learning algorithm can be thought of as a nonlinear map \( \mathcal{T} : \mathbb{R}^N \rightarrow \mathbb{R}^N \). The meaning of Eq. (4.1) is that the existing weight configuration \( w_n^k \) specifies the normal to a
hyperplane $\mathcal{H} = \{ x \in \mathbb{R}^N | w_n^k \cdot x + (-1)^{\lceil \frac{2^N+1}{2} \rceil} t = 0 \}$. Whether $y(w_n^k, x^j)$ is one or zero thus determines whether the perceptron currently "perceives" that the input vector $x^j$ should be on the one side of the hyperplane or the other. Thus learning in this context means the perceptron embodies a plane which properly separates the input vectors into two classes, which fall on either side of the plane. Indeed, there is a set of such hyperplanes, each one corresponding to a "rule" which the perceptron can learn. A rule specifies an output for each of the $2^N$ input vectors. $d^k = \{ d_j^k \}_{0 \leq j \leq 2^N - 1}$, $0 \leq k \leq 2^{2^N} - 1$ can be thought of as a $2^N$ component vector which describes a rule (or function) $k$ which the perceptron should learn, since its components are the desired outputs corresponding to each of the inputs. Since the perceptron’s output is either 0 or 1, there are thus $2^{2^N}$ possible rules to consider for the perceptron to learn.

At time $n = 0$ one of these $2^{2^N}$ rules is selected and the weight vectors are initialized to random values with components uniformly distributed in the $N$–dimensional hypercube centered at the origin, with faces orthogonal to the coordinate axes, and sides of length $2l$, in the weight space $\mathbb{R}^N$. For each time step $n$, each of the $2^N$ input vectors is presented in the cyclic online order. Weight vectors are changed according to the prescription:

$$w_{n+1}^k = w_n^k + a(d_j^k - y(w_n^k, x^j))x^j, \quad 0 \leq j \leq 2^N - 1,$$

where $a$ is the adaptation (learning) rate, and $j = n \mod 2^N$.

Of the $2^{2^N}$ possible functions to be learned, only a fraction will be learnable. From Eq. (4.2), the rule $k$ has been learned if the weights stop changing, which is equivalent to the condition that there exists $n_0 \in \mathbb{Z}^+$ such that $w_{n+1}^k = w_n^k$, for all $n \geq n_0$. In this case $y(w_n^k, x^j) = d_j^k$, $0 \leq j \leq 2^N - 1$, and thus the learning algorithm terminates. If the rule $k$ is not learnable, then the algorithm never terminates, i.e. $\lim_{n \to \infty} w_n^k$ does not exist. We discuss this matter further below in connection with the xor rule. There is also an equivalent theoretical notion of learnability, which we state for completeness. Let the input patterns be separated into two sets, according to whether the output should be 0 ($C^0$) or the output should be 1 ($C^1$). We say a rule $k$, $0 \leq k \leq 2^{2^N} - 1$ is learnable,
or "linearly separable" if there exists a number $\delta > 0$ and a weight (suppressing subscripts and superscripts momentarily) $w^* \in \mathbb{R}^N$ such that:

$$w^* \cdot x < -\delta, \text{ if } x \in C^0,$$

and

$$w^* \cdot x > \delta, \text{ if } x \in C^1.$$

Note that the definition of linear separability is equivalent to the existence of a hyperplane $\mathcal{H}^k \in \mathbb{R}^{N-1}$ which separates the two sets of input vectors according to whether the desired output is 0 or 1. Indeed, to see this one takes $w^*$ as the normal to the hyperplane, then the input vectors as points in $\mathbb{R}^N$ fall on either side of $\mathcal{H}^k$ by the equation for $\mathcal{H}^k: w^* \cdot x = 0$.

We now specialize to the case of $N = 2$, in order to derive exact formulas for the average convergence rates. A rule $k$ (boolean function) is identified by the vector of its four outputs $d^k = (d^k_0, d^k_1, d^k_2, d^k_3)$, corresponding to input vectors $(0, 0), (0, 1), (1, 0), \text{ and } (1, 1)$, respectively. For $N = 2$, 14 of 16 possible functions are linearly separable. By the symmetry of the patterns, the number of classes, based on the average number of steps to converge, is five. There are two attractors in class I ($d^3 = (0, 0, 0, 0)$ and $d^{15} = (1, 1, 1, 1)$), two attractors in class II ($d^1 = (0, 0, 0, 1)$ and $d^{14} = (1, 1, 1, 0)$), four attractors in class III ($d^2 = (0, 0, 1, 0), d^4 = (0, 1, 0, 0), d^{11} = (1, 0, 1, 1), \text{ and } d^{13} = (1, 1, 0, 1)$), four attractors in class IV ($d^3 = (0, 0, 1, 1), d^5 = (0, 1, 0, 1), d^{10} = (1, 0, 1, 0), \text{ and } d^{12} = (1, 1, 0, 0)$), and two attractors in class V ($d^7 = (0, 1, 1, 1)$ and $d^8 = (1, 0, 0, 0)$). We can see why the rules have the above class structure by the following symmetry considerations. If $d^k_1 = d^k_2$, we say the rule $d^k$ is symmetric, otherwise the rule is antisymmetric. If the rule is antisymmetric, then interchanging $d^k_1$ and $d^k_2$ gives a different rule (which we say is antisymmetric to the original rule) with equivalent dynamics, which yields a factor of 2. We say that a rule $k$ has equivalent weight dynamics to a rule $k$ if given initial conditions that are reflected across some line or point of symmetry, the average trajectories of the weights stay reflected in the same way for the two different...
rules. The components of the average trajectories, \( < w_{i,n} > \), are defined by:

\[
<w_{i,n} \equiv \frac{1}{4} \sum_{m=n}^{n+3} w_{i,m}, \text{ where } 1 \leq i \leq N.
\] (4.5)

In this case, it follows that if two different rules have equivalent dynamics, and the initial conditions are isotropic, then the weight vectors under both rules exhibit the same statistical convergence properties. We show in Appendix D (Appendix statement D.1) that if two rules \( k \) and \( \tilde{k} \) are antisymmetric, and if the initial conditions are antisymmetric (initial weight vector reflected across line \( w_1 = w_2 \)), then:

\[
\begin{pmatrix}
w_{k,1,4m}^1 \\
w_{k,1,4m}^2
\end{pmatrix}
= \begin{pmatrix}
w_{\tilde{k},2,4m}^1 \\
w_{\tilde{k},1,4m}^2
\end{pmatrix}, \quad m = 0, 1, \ldots
\] (4.6)

Thus two antisymmetric rules with antisymmetric initial conditions have trajectories which coincide every 4th step in the algorithm. In fact, according to statement D.1 of Appendix D, the average dynamics for antisymmetric rules \( k \) and \( \tilde{k} \) are antisymmetric given antisymmetric initial conditions:

\[
\begin{pmatrix}
<w_{k,1,n}^1 > \\
<w_{k,2,n}^1 > \\
<w_{k,1,n}^2 > \\
<w_{k,2,n}^2 >
\end{pmatrix}
= -\begin{pmatrix}
<w_{\tilde{k},1,n}^1 > \\
<w_{\tilde{k},2,n}^1 > \\
<w_{\tilde{k},1,n}^2 > \\
<w_{\tilde{k},2,n}^2 >
\end{pmatrix}, \quad n = 0, 1, \ldots
\] (4.7)

If two rules \( k \) and \( \tilde{k} \) are opposite (rule \( k \) is the opposite of rule \( \tilde{k} \) if \( d^k = d^{\tilde{k}} \)), then the weight dynamics will also be equivalent under the two rules. In particular, in Appendix D it is shown (Appendix statement D.3) that if two rules \( k \) and \( \tilde{k} \) are opposite and the corresponding initial conditions for the weights are reflected through the origin, then the average trajectories themselves will be reflected through the origin for the two rules, i.e.:

\[
\begin{pmatrix}
<w_{k,1,n}^1 > \\
<w_{k,2,n}^1 > \\
<w_{k,1,n}^2 > \\
<w_{k,2,n}^2 >
\end{pmatrix}
= -\begin{pmatrix}
<w_{\tilde{k},1,n}^1 > \\
<w_{\tilde{k},2,n}^1 > \\
<w_{\tilde{k},1,n}^2 > \\
<w_{\tilde{k},2,n}^2 >
\end{pmatrix}, \quad n = 0, 1, \ldots
\] (4.8)

Thus this yields an additional factor of two for the number of members of each rule class. Hence each class of symmetric rules, of which there are three, has
two members, and each class of antisymmetric rules has four members. This accounts for all 14 learnable of 16 possible rules.

The xor rule, \( d^6 = (0,1,1,0) \) and its opposite \((1,0,0,1)\) are not learnable, which can be seen by trying to separate by one line the points \((1,0)\) and \((0,1)\) in one class from \((0,0)\) and \((1,1)\) in the other. This also follows by a sufficient condition for non-learnability. Let \( x^i \) and \( x^j \) be two vectors in one of the classes to be separated, say \( C_0 \). It follows that if they are both on the same side of the separating hyperplane, the line \( L \) through these points must lie completely on that side of the hyperplane. It follows that the classification is unlearnable if some other line \( L' \) connecting two points in \( C_1 \) intersects the line \( L \). For the case of the xor rule, take \( x^2 = (0,1) \), and \( x^3 = (1,0) \), then the line \( L \) through \( x^2 \) and \( x^3 \) intersects the line \( L' \) connecting the two points \((0,0)\) and \((1,1)\) from the other class at the point \((1/2,1/2)\) so that the rule is not learnable. Thus there are three symmetric rule classes, with two elements in each class. This accounts for all 14 learnable of 16 possible rules. Note that if the distance between two randomly situated initial weight vectors were fixed at random values and this distance were observed over time for the perceptron attempting to learn the xor rule, this value would cycle periodically with various small periods. This indicates that the xor rule does not have a chaotic attractor.

Calculating the weight vectors for the five representatives of the convergent boolean functions, the solutions of the resulting linear inequalities are shown in Fig. (4.1). As expected, these figures, which are the "attractors" for the weight dynamics, correspond exactly to the attractors found by the perceptron convergence algorithm in Fig. (4.2). In Table 4.1 we give the linear inequalities for each of the five class representatives, corresponding to Fig. (4.1). In general, these inequalities for any of the 16 rules take the form:

\[
\begin{align*}
    w_2(-1)^d_1 &\leq t(-1)^d_1, \\
    w_1(-1)^d_2 &\leq t(-1)^d_2, \\
    w_2(-1)^d_3 &\leq (-w_1+t)(-1)^d_3.
\end{align*}
\]

From solving the linear inequalities for all 14 attractors we see that the attractors partition \( \mathbb{R}^2 \) twice as required for \( N = 2 \).
Figure 4.1: The solutions of the linear inequalities in Table 4.1, corresponding to the attractors of the five learnable classes for the single-layer perceptron. a) Class I representative (0,0,0,0), b) Class II representative (0,0,0,1), c) Class III representative (0,0,1,0), d) Class IV representative (0,0,1,1), e) Class V representative (0,1,1,1), Note that these regions coincide exactly with the attractors in weight space shown in Fig. (4.2).
Figure 4.2: The paths in weight space for 250 trials of the converging weight vectors. x: Beginning point of trajectory. *: End point of trajectory. a) Class I representative (0,0,0,0), $\mu = 24.47, \sigma = 22.19$, b) Class II representative (0,0,0,1), $\mu = 76.33, \sigma = 40.80$, c) Class III representative (0,0,1,0), $\mu = 43.59, \sigma = 34.34$, d) Class IV representative (0,0,1,1), $\mu = 41.44, \sigma = 31.04$, e) Class V representative (0,1,1,1), $\mu = 37.91, \sigma = 27.04$. By calculating the mean and standard deviation of the number of steps over these paths, we arrived at expressions 4.16-4.25.
### Table 4.1: Inequalities corresponding to attractors of Fig. (4.1), for the representatives of the five function classes. In Fig. (4.1) the threshold is set to \(-t = -1\).

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Attractor Inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0, 0, 0)</td>
<td>(w_2 \leq t, w_1 \leq t, w_2 \leq -w_1 + t)</td>
</tr>
<tr>
<td>(0, 0, 0, 1)</td>
<td>(w_2 \leq t, w_1 \leq t, w_2 \geq -w_1 + t)</td>
</tr>
<tr>
<td>(0, 0, 1, 0)</td>
<td>(w_2 \leq t, w_1 \geq t, w_2 \leq -w_1 + t)</td>
</tr>
<tr>
<td>(0, 0, 1, 1)</td>
<td>(w_2 \leq t, w_1 \geq t, w_2 \geq -w_1 + t)</td>
</tr>
<tr>
<td>(0, 1, 1, 1)</td>
<td>(w_2 \geq t, w_1 \geq t, w_2 \geq -w_1 + t)</td>
</tr>
</tbody>
</table>

#### 4.4 Convergence Rates

Consider the learnable functions (classifications) for the single-layer perceptron with \(N\) inputs. We fix the threshold at \(\pm t\), where \(t > 0\) is small. Then each convergent classification \(k\) corresponds to a finite or infinite attractor \(A_k \in \mathbb{R}^N\) for the weight dynamics. The attractor \(A_k\) is a convex set (a cone) formed by the intersection of two or more hyperplanes. Given an initial weight probability distribution with probability density function \(p(x)\), the expected number \(\mu_k\) of weight changes to converge is given by:

\[
\mu_k = \int_{\mathbb{R}^N} p(x) \eta(A_k, x) \, dx, \tag{4.10}
\]

where \(\eta(A_k, w)\) is essentially proportional to the length of the path from the initial weight vector \(w\) to the attractor \(A_k\). Similarly, the variance \(\sigma_k^2\) of the expected number of weight changes is given by:

\[
\sigma_k^2 = \int_{\mathbb{R}^n} p(x) \eta(A_k, x)^2 \, dx - \mu_k^2. \tag{4.11}
\]

To find \(\eta(A_k, x)\), we recall from Eq. (4.2) that the average trajectory of the weight vector is determined by the map:

\[
w_{n+1} = w_n + ac_R, \tag{4.12}
\]
where
\[ c_R = \frac{1}{2N} \sum_{j=1}^{2^N} (d^k_j - y(\langle w^k \rangle_R, x^j))x^j, \quad (4.13) \]
and \( \langle w^k \rangle_R \) denotes a range of weight vectors within the region \( R \) which produce the same output for \( y \). Thus, given that the weight vector’s path will be determined by the initial position of the weight vector \( x = w_0 \) and the rule \( k \), the form \( \eta(A_k, x) \) takes is:

\[ \eta(A_k, x) = \sum_{R \in \mathbb{R}^N} r_Rd_R, \quad (4.14) \]

where \( d_R \) is a distance computed from the direction \( c_R \) of the trajectory in \( R \) and the location of the next region, and \( r_R \) is a rate equal to the number of steps per unit distance travelled by the weight vector. The rate \( r_R \) can be determined from \( a \) and observing which values of \( j, \ 1 \leq j \leq 2^N \) produce weight changes in region \( R \), and it measures zigzagging on a finer level than that of the weights as they traverse the different regions. Since some trajectories travel close along the boundaries of the regions, the rate \( r_R \) was sometimes based on a best estimate of the trajectory at this finest level. In Appendix E we give an example calculation of Eq. (4.20), by using Eqs. (4.10) and (4.12).

In our investigation, \( N = 2 \), and we use a uniform probability distribution of initial weight vectors on a square of side length \( 2l \):

\[ p(x) = \frac{1}{4l^2} 1(|x| \leq l, |y| \leq l) \quad (4.15) \]

Computing \( \mu_k \) and \( \sigma_k^2 \) for \( k = 1 \) to 5 from Eq. (4.10) and 4.11 for the five function classes, we find:

\[ \mu_1(a, l, t) = \frac{22l^3 - 30l^2t + 30lt^2 - 49t^3}{48al^2} \quad (4.16) \]
\[ \sigma_1^2(a, l, t) = \frac{1}{2304a^2l^4} (356l^6 - 504l^5t + 804l^4t^2 + 980l^3t^3 - 5628l^2t^4 + 2940lt^5 - 2401t^6) \quad (4.17) \]
\[ \mu_2(a, l, t) = \frac{30l^3 - 30l^2t + 27lt^2 - 11t^3}{24al^2} \quad (4.18) \]
\[ \sigma_2^2(a, l, t) = \frac{1}{576a^2l^4} (204l^6 + 72l^5t - 864l^4t^2 + 1536l^3t^3 \]
\[ \quad - 1317l^2t^4 + 594lt^5 - 121l^6) \] (4.19)

\[ \mu_3(a, l, t) = \frac{103l^3 + 93l^2t + 97lt^2 + 7t^3}{192al^2} \] (4.20)

\[ \sigma_3^2(a, l, t) = \frac{1}{110592a^2l^4} (10725l^6 - 90018l^5t + 207435l^4t^2 \]
\[ \quad - 188788l^3t^3 + 44275l^2t^4 - 4074lt^5 - 147l^6) \] (4.21)

\[ \mu_4(a, l, t) = \frac{113l^3 + 3l^2t + 93lt^2 - 7t^3}{192al^2} \] (4.22)

\[ \sigma_4^2(a, l, t) = \frac{1}{38864a^2l^4} (8135l^6 + 2874l^5t + 5901l^4t^2 - 2336l^3t^3 \]
\[ \quad - 4119l^2t^4 + 1302lt^5 - 49l^6) \] (4.23)

\[ \mu_5(a, l, t) = \frac{44l^3 + 84l^2t + 42lt^2 - t^3}{96al^2} \] (4.24)

\[ \sigma_5^2(a, l, t) = \frac{1}{9216a^2l^4} (1424l^6 + 2976l^5t + 1056l^4t^2 \]
\[ \quad - 1112l^3t^3 - 720l^2t^4 + 84lt^5 - l^6) \] (4.25)

Following the logic of the above calculation, a generalization shows that the mean number of steps to converge for a linearly separable rule \( k \) for the general case of an arbitrary number of inputs \( N \) is a polynomial in \( \frac{t}{a} \):

\[ \mu_k(\frac{t}{a}; \frac{l}{a}) = \frac{1}{a} \sum_{j=0}^{N+1} a_j (\frac{t}{l})^j, \] (4.26)

treating \( \frac{l}{a} \) as a parameter, where the \( N + 1 \) coefficients can in principle be calculated. Thus by using an appropriate form of optimisation to calculate the coefficients \( a_j \) in Eq. (4.26), our method can be extended to yield the mean polynomial time convergence expressions for a perceptron with an arbitrary number \( N \) of inputs.

Note that the formulas we give here are valid within the range \( a < t < l \), which is a reasonable and normal ordering for these parameters. In particular, our simulations were performed taking the values \( a = .15, t = 1, \) and \( l = 10 \). If any of these parameters were to approach equality with any of the others, it is not expected that the above formulas would continue to be valid. Thus these results come with the caveat that they should only be used for "typical" ranges.
Table 4.2: Predicted and observed means and standard deviations of the number of steps to converge from Eqs. (4.16-4.25) and 10,000 trials. To compare note that e.g. the standard error of the mean for (0,0,0,0) would be $23.98/\sqrt{10,000} = .24$.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Theory Mean</th>
<th>St. Dev.</th>
<th>Simulation Mean</th>
<th>St. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,0,0)</td>
<td>26.74</td>
<td>24.61</td>
<td>25.77</td>
<td>23.98</td>
</tr>
<tr>
<td>(0,0,0,1)</td>
<td>75.72</td>
<td>39.67</td>
<td>77.50</td>
<td>40.06</td>
</tr>
<tr>
<td>(0,0,1,0)</td>
<td>39.33</td>
<td>12.05</td>
<td>44.39</td>
<td>35.58</td>
</tr>
<tr>
<td>(0,0,1,1)</td>
<td>39.66</td>
<td>31.97</td>
<td>40.24</td>
<td>32.45</td>
</tr>
<tr>
<td>(0,1,1,1)</td>
<td>36.68</td>
<td>28.89</td>
<td>37.05</td>
<td>29.22</td>
</tr>
</tbody>
</table>

4.5 Results

Fig. (4.3) shows the histograms of the number of steps to converge for the five pattern classes, using simulations of 10,000 trials. The corresponding means and variances as calculated from equations 4.16-4.25 are given in Table 4.2 (In the simulation and Table 4.2 we give the standard deviation, for its greater qualitative meaningfulness, instead of the variance.). We find that the simulation results agree very well with the values given by the calculated expressions for the means and variances. In particular, four of the five means calculated agree within a couple of standard errors of the mean (SEM) with the simulation results. Only one rule, (0,0,1,0) deviated significantly between predicted (39.33) and simulated (44.39) mean number of steps. This is most likely due to some error in the calculation of the number of steps along boundaries of the regions [c.f. Fig. (4.2)].

Fig. (4.2) shows the paths of the weight vectors for 250 trials. In some of these a path has a maximum of three directions, each of which is a straight line (though on a finer scale this is a zigzag). This is a consequence of the form taken by the piecewise constant vector field $c_R$ [Eq. (4.13)]. Only three of the input weight vectors are nonzero, thus there will be a maximum of three directions. The trajectories are piecewise linear since multiples of the input vectors are
Figure 4.3: The histograms of the five different classes for 10,000 trials are shown. a) Class I representative (0,0,0,0), $\mu = 25.77$, $\sigma = 23.98$, b) Class II representative (0,0,0,1), $\mu = 77.50$, $\sigma = 40.06$, c) Class III representative (0,0,1,0), $\mu = 44.39$, $\sigma = 35.58$, d) Class IV representative (0,0,1,1), $\mu = 40.24$, $\sigma = 32.45$, e) Class V representative (0,1,1,1), $\mu = 37.05$, $\sigma = 29.22$. Note that four classes have fairly similar shapes, having their distributions weighted towards the left, while the ‘and’ rule is centrally distributed. Trials which were already in the attractors at $n = 0$ were omitted for clarity.
always added to the weight vector (c.f. equation 4.12).

4.6 Discussion

From our method of calculation, it follows that the smaller the attractor, the slower in general the rate of convergence. This is illustrated by Table 4.1 and Fig. (4.1). On the other hand, one can interpret our results as showing that some inputs to the network result in "resonance" [4, 78] in the learning rate. In particular the class of functions corresponding to the outputs (0, 0, 1, 0) and (0, 1, 0, 0) were found to be resonant under this interpretation, if we disregard the "output always off" rule, (0, 0, 0, 0). In a previous study, the order of magnitude of convergence rate is given as $\sqrt{P/\epsilon} \log (1/\epsilon)$, where $P$ is the number of patterns and $\epsilon$ is the adaptation [28]. This gives for our values of $P$ and $\epsilon$, the number of learning steps as 21. As shown above, our results are more precise, and they distinguish between the different learning rules which the earlier studies do not.

Another improvement of our results over previous studies of perceptron learning performance is our closed form calculation of the variances of the convergence rates. Recent work in the field of computational complexity has shown that hard computational problems may better be tackled by combining several different approaches [31], thus detailed knowledge of learning performance of each component algorithm is crucial. It is interesting to note in connection with these findings of polynomial (in the variable $t$) time convergence rates on average the results for $P$ complexity of single-layer perceptrons learning linearly separable rules, and $NP$–Complete complexity of two-layer perceptrons [11]. Indeed, our formulas suggest that the problem of a single-layer perceptron converging has polynomial complexity, by direct computation of such polynomials (Since our expressions give only averages we cannot claim they guarantee polynomial time convergence.). This dynamic between the single-layer perceptron polynomial time convergence versus the two-layer perceptron nondeterministic polynomial time convergence evokes a recent result which showed that $NP$-complete problems exhibit phase boundaries [46] away from which the problems become easier. Thus (considering a quasi-continuous case) inputs which transi-
tion from the 'xor' rule to the 'and' rule may be seen to undergo an analogous phase transition in difficulty. We therefore suggest that a greater understanding of the relation between $P$ and $NP$-complete computational complexity classifications might be obtained by generalizing our computational approach for two-layer neural networks, though it is not clear to us at present how to proceed in this undertaking.

By the same reasoning which led to the result that inequalities (4.9) give 14 attractors which partition $\mathbb{R}^2$ twice, it follows that a similar (though much larger) set of inequalities corresponding to the linearly separable boolean functions of a perceptron with $N$ inputs partitions $\mathbb{R}^N \oplus \mathbb{R}^N$. In particular, the perceptron must solve the system:

\[
\begin{align*}
    a_{11}w_1 + a_{12}w_2 + \ldots + a_{1N}w_N & \geq t, \\
    a_{21}w_1 + a_{22}w_2 + \ldots + a_{2N}w_N & \geq t, \\
    \vdots & \vdots \\
    a_{k1}w_1 + a_{k2}w_2 + \ldots + a_{kN}w_N & \geq t, \\
    a_{(k+1)1}w_1 + a_{(k+1)2}w_2 + \ldots + a_{(k+1)N}w_N & \leq t, \\
    \vdots & \vdots \\
    a_{2N1}w_1 + a_{2N2}w_2 + \ldots + a_{2N}w_N & \leq t,
\end{align*}
\]

where $0 \leq k \leq 2^N$, $-t$ is the numerical value of the threshold (for the higher order rules the threshold will be $t$), and the inequalities were rearranged so that all $\geq$ appear before any $\leq$, and $a_{ij} \in \{0,1\}$, $1 \leq i \leq 2^N$, $1 \leq j \leq N$. As with a system of $2^N$ linear equations in $N$ unknowns, there will either be an infinite number of solutions to this system, or no solution. In this connection there is a useful question to ask: is there a deterministic calculation which can always establish, as a function of $k$ and the coefficients $a_{ij}$, whether or not the system of inequalities (4.27) (and thus could the perceptron learn the corresponding rule) has a solution? We don’t have an answer for this, though we suggest that the answer would be yes. Nevertheless we can see from the above generalization that the set of attractors for the single-layer perceptron with $N$ inputs is isomorphic to $\mathbb{R}^N \oplus \mathbb{R}^N$. This leads to the question of how many separate regions into
which $\mathbb{R}^N$ can be divided.

In the general case of a perceptron with $N$ inputs, we have a situation of hyperplanes which partition $\mathbb{R}^N$ into regions in which the weight dynamics follows a straight line (see Eq. (4.12)]. If $N = 2$, then the hyperplanes are lines, and they partition $\mathbb{R}^2$ in the general case into 7 regions. For $N = 3$, the hyperplanes are planes in $\mathbb{R}^3$, and in this case we found that they divide $\mathbb{R}^3$ into 26 regions, as there were 26 attractors. It is interesting to speculate on how many regions hyperplanes in $\mathbb{R}^N$ can divide $\mathbb{R}^N$ into in the most general case, since this number would give an upper bound on the number of rules which a single-layer perceptron with $N$ inputs could learn. A simple counting argument shows that in $\mathbb{R}^2$ there are again 7 such regions. In this connection a result by Polya [51] gives as 64 the number of regions which planes divide $\mathbb{R}^3$ up into in the most general case, which indicates that perceptrons do not partition space in the most general way.

Note that for the general case of $\mathbb{R}^N$, our sufficient condition for non-learnability of a classification can be restated in terms of hyperplanes. In particular, let $\{P_1, \ldots, P_r\}$ and $\{Q_1, \ldots, Q_s\}$ be points in the classes $C_0$ and $C_1$ and denote their respective convex hulls by $S_0$ and $S_1$. Then if $S_0 \cap S_1 \neq \emptyset$, the classification is not learnable.

### 4.7 Conclusion

Our main result is the derivation of exact averages and variances of the convergence rates for $N = 2$ perceptrons, and the general polynomial form Eq. (4.26) of the average convergence rates for perceptrons with $N$ input vectors. Possible future work might, beyond extending these results to include multiperceptrons as mentioned above, include algorithms or methods for exact calculations of the coefficients for the polynomials given in Eq. (4.26). We suggest that similar mean convergence rate results may also be obtainable by generalizing this approach to other types of neural networks frequently used in practice. I.e., one might begin by calculating the mean convergence rate for a smallest possible network of a given type following a method similar to ours used to derive
Eqs. (4.16-4.25), then generalizing to arbitrary $N$ through the use of energy or information optimization to find the unknown coefficients in an expression analogous to Eq. (4.26). Such precise specifications for learning performance are essential as hardwire versions of neural networks are implemented in various mission-critical applications. For instance, it has been shown that the perceptron learning rule can be used as a "local learning rule" in Hopfield-like associative memory networks [20]. In addition, bottom-up self-assembly of molecular nanowires [1] holds promise as technologies for which neural network paradigms could be applied to advantage. Indeed, it has been shown that agglomerations of conducting particles self-assemble, form electrical connections, and exhibit Hebbian learning through the principle of minimum resistance [68, 21, 35]. Hence, higher precision insights into perceptron learning rates—gained by analyzing the geometric nature of their attractors—may have numerous practical applications.
Chapter 5

Conclusions

Desalination graphs as models of flow networks embedded in a diffusive saline medium were generated with iterated function systems. The diffusion equations were solved by analogy to electrostatics and put into dimensionless form. Solutions showing the optimal graphs, as well as a discussion of the performance of asymmetric graphs were presented for the case of constant pressure differences at the absorbers, which is experimentally the most accessible case. Subsequently extensions of this research, including the consideration of other boundary conditions and possible experiments, were presented. Finally, in the context of smart materials and self-adjusting systems, the speed of convergence of a feed-forward perceptron network doing online learning was studied. General polynomial expressions for the mean and variance of the number of steps to converge for weights initially uniformly distributed were given.
Appendix A

Iterated Function System
Linear Transformation

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
-\rho \cos \alpha & \pm \rho \sin \alpha & 0 & \rho \cos \alpha + 1 & \mp \rho \sin \alpha & 0 \\
\mp \rho \sin \alpha & -\rho \cos \alpha & 0 & \pm \rho \sin \alpha & \rho \cos \alpha + 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}, \quad (A.1)
\]
Appendix B

Expansion about $c_i' = \frac{\Delta}{RT}$

We want to apply Taylor's rule to the function

$$f_i(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_G) = \hat{c}_i + \frac{k}{\sqrt{4\pi 2^G}} \hat{c}_i(1 + \xi \hat{c}_i) + \frac{k\beta}{4\pi 2^G} \sum_{j \neq i} \hat{c}_j(1 + \xi \hat{c}_j). \quad (B.1)$$

The first order Taylor series approximation to $f_i(\hat{c})$ about $\hat{c} = 0$ is just:

$$f_i(\hat{c}) = f_i(0) + \sum_{j=1}^{2^G} \frac{\partial f_i}{\partial \hat{c}_j}(0) \hat{c}_j. \quad (B.2)$$

Then, since

$$f_i(0) = 0,$$
$$\frac{\partial f_i}{\partial \hat{c}_i}(0) = 1 + \frac{k}{\sqrt{4\pi 2^G}},$$
$$\frac{\partial f_i}{\partial \hat{c}_j}(0) = \frac{k\beta}{4\pi 2^G |\mathbf{v}_{G,j} - \mathbf{v}_{G,i}|}, \quad (B.3)$$
$$\frac{\partial^2 f_i}{\partial \hat{c}_i^2}(0) = \frac{2k\xi}{\sqrt{4\pi 2^G}},$$
$$\frac{\partial^2 f_i}{\partial \hat{c}_j^2}(0) = \frac{2k\beta\xi}{4\pi 2^G |\mathbf{v}_{G,j} - \mathbf{v}_{G,i}|}, \quad (B.4)$$
$$\frac{\partial^2 f_i}{\partial \hat{c}_k \partial \hat{c}_j}(0) = 0,$$

we find Eq. (2.20):

$$f_i(\hat{c}_i) \simeq (1 + \frac{k}{\sqrt{4\pi 2^G}})\hat{c}_i + \frac{k\beta}{4\pi 2^G} \sum_{j \neq i} \frac{\hat{c}_j}{|\mathbf{v}_{G,j} - \mathbf{v}_{G,i}|} = 1, \quad (B.5)$$

with the neglected nonlinear term being

$$\frac{1}{2} \sum_{k=1}^{2^G} \sum_{j=1}^{2^G} \frac{\partial^2 f_i}{\partial \hat{c}_k \partial \hat{c}_j}(0) \hat{c}_k \hat{c}_j = \frac{k\xi}{\sqrt{4\pi 2^G}} \hat{c}_i^2 + \frac{k\beta\xi}{4\pi 2^G} \sum_{j \neq i} \frac{\hat{c}_j^2}{|\mathbf{v}_{G,j} - \mathbf{v}_{G,i}|}. \quad (B.6)$$
Appendix C

Sufficient Condition for Solution

Eq. (2.20) has a unique solution if the linear system is nonsingular. Writing the matrix $A$ of coefficients of the system,

$$A = \begin{pmatrix}
1 + k & \frac{R_k}{|v_{G,2} - v_{G,1}|} & \cdots & \frac{R_k}{|v_{G,2} - v_{G,1}|} \\
\frac{R_k}{|v_{G,1} - v_{G,2}|} & 1 + k & \cdots & \frac{R_k}{|v_{G,2} - v_{G,1}|} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{R_k}{|v_{G,1} - v_{G,2}|} & \cdots & \cdots & 1 + k
\end{pmatrix}, \quad (C.1)
$$

By the Gershgorin circle theorem, as shown originally by L. Levy[74], $A$ will be invertible if it is diagonally dominant. I.e., it suffices that

$$l \frac{1+k}{k} > R_k \sum_{i \neq j} \frac{1}{|v_{G,j} - v_{G,i}|} \quad (C.2)$$

holds for all $i = 1, \ldots, 2^G$. Let

$$\gamma_i = \sum_{i \neq j} \frac{1}{|v_{G,j} - v_{G,i}|} \quad (C.3)$$

then define $\gamma_{\max} = \max_{i=1,\ldots,2^G} \gamma_i$. Since Eq. C.2 is then satisfied if

$$\frac{l(1+k)}{R_\alpha k} > \gamma_{\max}, \quad (C.4)$$
we thus arrive at the form:

\[
\max_{i=1,\ldots,2^G} \{ \sum_{i\neq j} \frac{1}{|v_{G,j} - v_{G,i}|} \} < \frac{l(1+k)}{k} \sqrt{\frac{4\pi 2^G}{A_n}}. \quad (C.5)
\]
Appendix D

Equivalent Dynamics

We prove that reflected weight trajectories are close approximations to the original trajectories. This is true for trajectories reflected through the line \( w_1 = w_2 \), which is the content of statement D.1, and for trajectories reflected through the origin, as shown by statement D.2. These two statements, combined with the definition of an average trajectory, yield statement D.3, which shows that the appropriately reflected average dynamics coincides for antisymmetric and opposite rules. Thus statement D.3 gives theoretical justification for only considering 5 of the possible 14 learnable rules in our analysis, since other members of the respective classes have equivalent dynamics.

D.1 (Antisymmetric Rules) Let \( k \) and \( \bar{k} \) be antisymmetric rules, then if

\[
\begin{pmatrix}
w^k_{1,0} \\
w^k_{2,0}
\end{pmatrix} = \begin{pmatrix}
w^\bar{k}_{2,0} \\
w^\bar{k}_{1,0}
\end{pmatrix},
\]  
(D.1)

\[
\begin{pmatrix}
w^k_{1,4m} \\
w^k_{2,4m}
\end{pmatrix} = \begin{pmatrix}
w^\bar{k}_{2,4m} \\
w^\bar{k}_{1,4m}
\end{pmatrix}, \quad m = 0, 1, \ldots
\]  
(D.2)

This statement says that if the initial conditions of the weights are antisymmetric (reflected through the line \( w_1 = w_2 \)), then the weight trajectories of corresponding antisymmetric rules will be antisymmetric (reflected through \( w_1 = w_2 \)).

Proof of statement D.1

We give a proof by induction. Note that since the statement is true for \( m = 0 \)
by assumption, it suffices to prove only the induction step. Thus we suppose the statement is true for some \( m \). Then we show that statement is true for \( m + 1 \), so that by the induction hypothesis, the statement will be proven. Applying Eqs. (4.1) and (4.2) repeatedly we find for rule \( k \):

\[
\begin{pmatrix}
  w^k_{1,4m+1} \\
  w^k_{2,4m+1}
\end{pmatrix} =
\begin{pmatrix}
  w^k_{1,4m} \\
  w^k_{2,4m}
\end{pmatrix},
\]

\( (D.3) \)

\[
\begin{pmatrix}
  w^k_{1,4m+2} \\
  w^k_{2,4m+2}
\end{pmatrix} =
\begin{pmatrix}
  w^k_{1,4m} + a(d^k_1 - \Theta(w^k_{2,4m} - t)) \\
  w^k_{2,4m}
\end{pmatrix}
\begin{pmatrix}
  0 \\
  1
\end{pmatrix}
\]

\( (D.4) \)

\[
\begin{pmatrix}
  w^k_{1,4m+3} \\
  w^k_{2,4m+3}
\end{pmatrix} =
\begin{pmatrix}
  w^k_{1,4m+2} + a(d^k_2 - \Theta(w^k_{2,4m+2} - t)) \\
  w^k_{2,4m+2}
\end{pmatrix}
\begin{pmatrix}
  1 \\
  0
\end{pmatrix}
\]

\( (D.5) \)

\[
\begin{pmatrix}
  w^k_{1,4m+4} \\
  w^k_{2,4m+4}
\end{pmatrix} =
\begin{pmatrix}
  w^k_{1,4m+3} + a(d^k_3 - \Theta(w^k_{1,4m+3} + w^k_{2,4m+3} - t)) \\
  w^k_{2,4m+3} + a(d^k_3 - \Theta(w^k_{1,4m+3} + w^k_{2,4m+3} - t))
\end{pmatrix}
\]

\( (D.6) \)

For rule \( \hat{k} \) we have by an analogous series of calculations,

\[
\begin{pmatrix}
  w^\hat{k}_{1,4m+3} \\
  w^\hat{k}_{2,4m+3}
\end{pmatrix} =
\begin{pmatrix}
  w^\hat{k}_{2,4m} + a(d^\hat{k}_1 - \Theta(w^\hat{k}_{2,4m} - t)) \\
  w^\hat{k}_{1,4m} + a(d^\hat{k}_2 - \Theta(w^\hat{k}_{1,4m} - t))
\end{pmatrix}
\]

\( (D.7) \)

\[
\begin{pmatrix}
  w^\hat{k}_{2,4m+3} \\
  w^\hat{k}_{1,4m+3}
\end{pmatrix} =
\begin{pmatrix}
  w^\hat{k}_{2,4m+3} + a(d^\hat{k}_3 - \Theta(w^\hat{k}_{1,4m+3} + w^\hat{k}_{2,4m+3} - t)) \\
  w^\hat{k}_{1,4m+3} + a(d^\hat{k}_3 - \Theta(w^\hat{k}_{1,4m+3} + w^\hat{k}_{2,4m+3} - t))
\end{pmatrix}
\]

\( (D.8) \)

where use was made of the antisymmetry of the initial conditions and of the rules \( k \) and \( \hat{k} \), and Eq. (D.7). Then, for \( 4(m + 1) = 4m + 4 \), we have, again by
antisymmetry of the rules:

\[
\begin{pmatrix}
w_{k,1,4m+4} \\
w_{k,2,4m+4}
\end{pmatrix}
= \begin{pmatrix}
w_{k,2,4m+3} + a(d_k^1 - \Theta(w_{k,2,4m+3} + w_{k,1,4m+3} - t)) \\
w_{k,1,4m+3} + a(d_k^1 - \Theta(w_{k,2,4m+3} + w_{k,1,4m+3} - t))
\end{pmatrix}
\]
\[
= \begin{pmatrix}
w_{k,2,4m+4} \\
w_{k,1,4m+4}
\end{pmatrix},
\]

(D.11)

which verifies Eq. (D.2) by the induction hypothesis.

**D.2 (Opposite Rules)** Let \( k \) and \( \tilde{k} \) be opposite rules, then if

\[
\begin{pmatrix}
w_{k,1,0} \\
w_{k,2,0}
\end{pmatrix}
= -\begin{pmatrix}
w_{\tilde{k},1,0} \\
w_{\tilde{k},2,0}
\end{pmatrix},
\]

(D.12)

\[
\begin{pmatrix}
w_{k,1,4m} \\
w_{k,2,4m}
\end{pmatrix}
= -\begin{pmatrix}
w_{\tilde{k},1,4m} \\
w_{\tilde{k},2,4m}
\end{pmatrix}, \quad m = 0, 1, \ldots
\]

(D.13)

This statement says that if the initial conditions of the weights for two rules are opposite (of opposite sign), then the weight trajectories of corresponding opposite rules will be opposite (reflected through the origin).

**Proof of statement D.2**

We again give a proof by induction. As for statement D.1, the statement is true for \( m = 0 \) by assumption, so it suffices to prove only the induction step. Thus suppose the statement is true for some \( m \). Applying Eqs. (4.1) and (4.2) repeatedly we find for rule \( k \) Eqs. (D.7) and (D.8). For rule \( \tilde{k} \) we have:

\[
\begin{pmatrix}
w_{\tilde{k},1,4m+1} \\
w_{\tilde{k},2,4m+1}
\end{pmatrix}
= -\begin{pmatrix}
w_{k,1,4m} \\
w_{k,2,4m}
\end{pmatrix},
\]

(D.14)

\[
\begin{pmatrix}
w_{\tilde{k},1,4m+2} \\
w_{\tilde{k},2,4m+2}
\end{pmatrix}
= -\begin{pmatrix}
w_{k,1,4m} \\
w_{k,2,4m}
\end{pmatrix} + a(d_k^1 - \Theta(-w_{k,2,4m} + t)) \begin{pmatrix} 0 \\ 1 \end{pmatrix}
\]

\[
= \begin{pmatrix}
-w_{k,1,4m} \\
-w_{k,2,4m} + a(d_k^1 - \Theta(-w_{k,2,4m} + t))
\end{pmatrix},
\]

(D.15)
\[
\begin{pmatrix}
  w_{1,4m+3} \\
  w_{2,4m+3}
\end{pmatrix} = \begin{pmatrix}
  u_{1,4m+2} \\
  u_{2,4m+2}
\end{pmatrix} + a(d_k^2 - \Theta(w_{1,4m+2}^k + t)) \begin{pmatrix}
  1 \\
  0
\end{pmatrix}
\]
\[
\begin{pmatrix}
  w_{1,4m+2} \\
  w_{2,4m+2}
\end{pmatrix} = \begin{pmatrix}
  w_{1,4m+2}^k + a(d_k^2 - \Theta(w_{1,4m+2}^k + t)) \\
  w_{2,4m+2}
\end{pmatrix}
\]
\[
(D.16)
\]
\[
\begin{pmatrix}
  -w_{1,4m}^k + a(d_k^2 - \Theta(-w_{1,4m}^k + t)) \\
  -w_{2,4m}^k + a(d_k^2 - \Theta(-w_{2,4m}^k + t))
\end{pmatrix},
\]
\[
(D.17)
\]
\[
\begin{pmatrix}
  -w_{1,4m}^k + a(d_k^2 - \tilde{\Theta}(w_{1,4m}^k - t)) \\
  -w_{2,4m}^k + a(d_k^2 - \tilde{\Theta}(w_{2,4m}^k - t))
\end{pmatrix},
\]
\[
(D.18)
\]
\[
\begin{pmatrix}
  -w_{1,4m}^k + a(d_k^2 - \Theta(w_{1,4m}^k - t)) \\
  -w_{2,4m}^k + a(d_k^2 - \Theta(w_{2,4m}^k - t))
\end{pmatrix},
\]
\[
(D.19)
\]

thus by Eq. (D.7),
\[
\begin{pmatrix}
  w_{1,4m+3}^k \\
  w_{2,4m+3}^k
\end{pmatrix} = \begin{pmatrix}
  -w_{1,4m+3}^k \\
  -w_{2,4m+3}^k
\end{pmatrix}.
\]
\[
(D.20)
\]

When \( n = 4(m + 1) = 4m + 4 \),
\[
\begin{pmatrix}
  w_{1,4m+4}^k \\
  w_{2,4m+4}^k
\end{pmatrix} = \begin{pmatrix}
  -w_{1,4m+3}^k + a(d_3^2 - \Theta(-w_{2,4m+3}^k - w_{1,4m+3}^k + t)) \\
  -w_{2,4m+3}^k + a(d_3^2 - \Theta(-w_{2,4m+3}^k - w_{1,4m+3}^k + t))
\end{pmatrix}
\]
\[
(D.21)
\]

using Eqs. (D.20) and (D.3).

**D.3 (Average Dynamics)** If \( k \) and \( \tilde{k} \) are antisymmetric or opposite rules,
then the average dynamics of \( k \) and \( \tilde{k} \) are equivalent. I.e.,

\[
\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix}
= \begin{pmatrix}
< w_{2,n}^k > \\
< w_{1,n}^k >
\end{pmatrix}, \quad n = 0, 1, \ldots, \tag{D.22}
\]

for antisymmetric rules, or

\[
\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix}
= - \begin{pmatrix}
< \tilde{w}_{1,n}^\tilde{k} > \\
< \tilde{w}_{2,n}^\tilde{k} >
\end{pmatrix}, \quad n = 0, 1, \ldots, \tag{D.23}
\]

for opposite rules, where

\[
< w_{i,n} > := \frac{1}{4} \sum_{m=n}^{n+3} w_{i,m}, \quad 1 \leq i \leq N. \tag{D.24}
\]

**Proof of statement D.3**

We again give a proof by induction. Suppose first that \( k \) and \( \tilde{k} \) are antisymmetric rules. Then from the induction assumption

\[
\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix}
= \begin{pmatrix}
< w_{2,n}^k > \\
< w_{1,n}^k >
\end{pmatrix}, \tag{D.25}
\]

we have:

\[
\begin{pmatrix}
< w_{1,n+1}^k > \\
< w_{2,n+1}^k >
\end{pmatrix}
= \begin{pmatrix}
\frac{1}{4} \sum_{m=n+1}^{n+4} w_{1,m}^k \\
\frac{1}{4} \sum_{m=n+1}^{n+4} w_{2,m}^\tilde{k}
\end{pmatrix}, \tag{D.26}
\]

\[
= \begin{pmatrix}
< w_{1,n+4}^k > - w_{1,n}^k \\
< w_{2,n+4}^k > - w_{2,n}^k
\end{pmatrix} + \frac{1}{4} \begin{pmatrix}
w_{1,n+4}^k - w_{1,n}^k \\
w_{2,n+4}^k - w_{2,n}^k
\end{pmatrix}, \tag{D.27}
\]

\[
= \begin{pmatrix}
< w_{2,n}^k > - w_{1,n}^k \\
< w_{2,n}^k > + \tilde{\Delta}
\end{pmatrix}. \tag{D.28}
\]
Then, since

\[
\begin{pmatrix}
  \langle w_{2,n+1}^k \rangle \\
  \langle w_{1,n+1}^k \rangle
\end{pmatrix}
= \begin{pmatrix}
  \langle w_{2,n}^k \rangle \\
  \langle w_{1,n}^k \rangle
\end{pmatrix} + \frac{1}{4} \begin{pmatrix}
  w_{2,n+4}^k - w_{2,n}^k \\
  w_{1,n+4}^k - w_{1,n}^k
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  \langle w_{2,n}^k \rangle \\
  \langle w_{1,n}^k \rangle
\end{pmatrix} + \Delta,
\]

(D.29)

it suffices to verify that

\[
4(\tilde{\Delta} - \Delta) = \begin{pmatrix}
  w_{1,n}^k - w_{1,n}^k - w_{2,n+4}^k + w_{2,n}^k \\
  w_{2,n+4}^k - w_{2,n}^k - w_{1,n+4}^k + w_{1,n}^k
\end{pmatrix}
\]

\[
= 0.
\]

(D.30)

Proceeding case by case for the values of \( n \), we have first the case \( n = 4m \), which yields:

\[
4(\tilde{\Delta} - \Delta) = \begin{pmatrix}
  w_{1,4(m+1)}^k - w_{1,4(m+1)}^k - (w_{2,4m}^k - w_{2,4m}^k) \\
  w_{2,4(m+1)}^k - w_{2,4(m+1)}^k - (w_{1,4m}^k - w_{1,4m}^k)
\end{pmatrix}
\]

\[
= 0.
\]

(D.31)

(D.32)

by appendix statement D.1. When \( n = 4m + 1 \), \( 4(\tilde{\Delta} - \Delta) = 0 \) by Eq. (D.3). If \( n = 4m + 2 \), then

\[
4(\tilde{\Delta} - \Delta)_1 = w_{1,4m+6}^k - w_{2,4m+2}^k - w_{2,4m+6}^k + w_{2,4m+2}^k
\]

\[
= w_{2,4m+4}^k - w_{2,4m+2}^k - w_{2,4m+4}^k + w_{2,4m+2}^k
\]

\[
= 0,
\]

(D.35)

(D.36)

(D.37)

and

\[
4(\tilde{\Delta} - \Delta)_2 = w_{1,4m+4}^k + a(d_2^k - \theta(w_{1,4m+4}^k - t)) - w_{1,4m}^k
\]

\[
- a(d_2^k - \theta(w_{1,4m}^k - t)) - w_{1,4m+4}^k + w_{1,4m}^k
\]

\[
= a(\theta(w_{1,4m}^k - t) - \theta(w_{1,4m+4}^k - t))
\]

\[
\cong 0
\]

(D.38)

(D.39)

(D.40)
where Eq. (D.5) was used (The second component is bounded by $|a| \ll 1$ when nonzero, and is only nonzero when the weight trajectory crosses some regional boundary.). For $n = 4m + 3$, $4(\tilde{\Delta} - \Delta) = 0$ by Eq. (D.10). Thus if Eq. (D.22) is true for $n$, it is true for $n + 1$. It is true in particular for $n = 0$, since this is the case $n = 4m$. Hence by the induction hypothesis, Eq. (D.22) is verified for all $n$.

Now suppose that $k$ and $\tilde{k}$ are opposite rules. Assume the statement true for $n$, so that

$$\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix}
= -\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix}.
$$

(D.41)

Then

$$\begin{pmatrix}
< w_{1,n+1}^k > \\
< w_{2,n+1}^k >
\end{pmatrix}
= -\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix} + \tilde{\Delta}, \text{ and (D.42)}$$

$$-\begin{pmatrix}
< w_{1,n+1}^k > \\
< w_{2,n+1}^k >
\end{pmatrix}
= -\begin{pmatrix}
< w_{1,n}^k > \\
< w_{2,n}^k >
\end{pmatrix} + \Delta, \text{ where (D.43)}$$

$$\tilde{\Delta} = \frac{1}{4}\begin{pmatrix}
w_{1,n+4}^k - w_{1,n}^k \\
w_{2,n+4}^k - w_{2,n}^k
\end{pmatrix}, \text{ and (D.44)}$$

$$\Delta = \frac{1}{4}\begin{pmatrix}
w_{1,n}^k - w_{1,n+4}^k \\
w_{2,n}^k - w_{2,n+4}^k
\end{pmatrix}. \text{ (D.45)}$$

We then proceed as for the antisymmetric rules by considering cases on $n$. When $n = 4m$, $4m + 1$, or $4m + 3$, we find $\tilde{\Delta} - \Delta = 0$ using appendix statement D.2, (D.14), or (D.20) respectively, by exactly analogous calculations as for the antisymmetric rules. If $n = 4m + 2$, then by (D.5) and (D.15) we get:

$$4(\tilde{\Delta} - \Delta)_1 = w_{1,4m+6}^k - w_{1,4m+2}^k - w_{1,4m+2}^k + w_{1,4m+6}^k \quad \text{(D.46)}$$

$$= -w_{1,4m+4}^k + w_{1,4m}^k - w_{1,4m}^k + w_{1,4m+4}^k \quad \text{(D.47)}$$

$$= 0, \text{ and (D.48)}$$
\[ 4(\bar{\Delta} - \Delta) = -w_{2,4m+4}^k + a(\tilde{d}_1^k - \theta(w_{2,4m+4}^k + t)) + w_{2,4m}^k \]
\[ - a(d_1^k - \theta(-w_{2,4m}^k + t)) - w_{2,4m}^k - a(d_1^k - \theta(w_{2,4m}^k - t)) \]
\[ + w_{2,4m+4}^k + a(d_1^k - \theta(w_{2,4m+4}^k - t)) \]
\[ = a(d_1^k - \hat{\theta}(w_{2,4m+4}^k - t)) - a(\tilde{d}_1^k - \hat{\theta}(w_{2,4m}^k - t)) \]
\[ - a(d_1^k - \theta(w_{2,4m}^k - t)) + a(d_1^k - \theta(w_{2,4m+4}^k - t)) \]
\[ = 0. \]

Since \( n = 0 \) is included in the case \( n = 4m \), this completes the induction for the opposite rules and thus the proof of statement D.3.
Appendix E

Example Calculation

In this appendix, we sketch the calculation of Eq. (4.20), using Eqs. (4.10) and (4.12). In Fig. (E.1) is shown the seven different regions to consider. If the initial weight value is in region $A_6$, then all the inequalities in the third line of Table (4.1) are satisfied, so that the weight vector is already in the attractor. If however the weight vector is in one of the other regions, then it will follow its trajectory to the attractor ($A_6$) as shown in Fig. (4.2c). Thus in order to find the integral in Eq. (4.10), the length of each possible trajectory is calculated. First each possible trajectory is determined to yield Eq. (4.12). Then Eq. (4.12) is used to compute the component distances $d_R$ in Eq. (4.14), in each segment of the trajectory. Finally, the rates $r_R$ in Eq. (4.14), which give the number of steps per unit distance travelled by the weight vector, are calculated. In Table (E.1) we give the details which were used to write down the integral for Eq. (4.10). Putting these details together, the integral for Eq. (4.10) was found to be:
Figure E.1: Regions $A_1 - A_7$ used for calculating Eq. (4.20) from Eqs. (4.10) and (4.12). The dashed lines corresponding to $w_1 = a$, $w_2 = a$, and $w_2 = -w_1 + a$ partition the initial weight vector space into the seven regions $A_1 - A_7$.

Table E.1: Details utilized to determine all possible trajectories for weight vectors initially in the different regions $A_1 - A_7$. The condition violated was used to select the weight change, which in turn gave the slope of the trajectory segment. By determining the distance $d_R$ of the complete trajectory from arbitrary starting location to the attractor $A_6$, and multiplying by the rate $r_R$ for each different segment, the integral of Eq. (E.1) was written down.
\( \mu_3(a, l, t) = \frac{1}{4al^2} \{ \)

\[ + \int_0^t \int_{t-y}^t [2(x + y - t) + 3(t - x)] \, dx \, dy \\
+ \int_0^t \int_{t-y}^{t+y} [(x - t) + 2(t + y - x)] \, dx \, dy \\
+ \int_0^t \int_{t+y}^t (x/2 - t/2 + y/2) \, dx \, dy \\
+ \int_{t-l}^0 \int_{t+y}^t (x/2 - t/2 + y/2) \, dx \, dy \\
+ \int_0^t \int_{-l}^{t-y} [(t - y - x) + 3y] \, dx \, dy \\
+ \int_0^t \int_{-l}^t (t - x) \, dx \, dy \ \\
+ \int_{t-l}^t \int_{-l-y}^t [2(y - t) + (t - x - y) + 3t] \, dx \, dy \\
+ \int_{t}^t \int_{t-y}^{t-y} [3/2(x + y - t) + (-2x) + 3t] \, dx \, dy \\
+ \int_{t}^t \int_{t-y}^t [3/2(y - t) + 2x + 3(t - x)] \, dx \, dy \\
+ \int_{t}^t \int_{3t/2+y/2}^{t/2+y/2} [2(x - t) + 4/3(t - 2x + y) + 2t] \, dx \, dy \\
+ \int_{t}^t \int_{t/2+y/2}^{3t/2+y/2} [(y - t) + (-t/2 + x - y/2) + 2(3t/2 - x + y/2)] \, dx \, dy \\
+ \int_{t}^t \int_{3t/2+y/2}^t [(y - t) + (t/4 + x/2 - y/4)] \, dx \, dy. \} \]

Integrating and simplifying this expression with Mathematica then yields Eq. (4.20)
Appendix F

Code for Desalination Study

function desal10( G, k, beta)
    
    % calculates solutions to desalination with fractal absorbers problem for
    % boundary condition of constant pressure difference, for case of bi-turtle
    % graphs usage: desal10 G k beta
    % G: generations of turtle graph
    % k: dimensionless pressure
    % beta: area-length scaling factor
    % desal6 adds to desal5 checking of error made in linearization

    % desal8 does calculation based only on dimensionless parameters as in
    % 12–10–09 version of paper
    % desal9 omits reference to dimensional variable so script only solves the
    % dimensionless equation (20) in 12–23–09 version of paper
    % desal10 solves systems over more refined range to improve shapes of plots

    format('long')

    if (nargin ~= 3)
        help desal10;
        return;
    else
        G = str2num(G);
        k = str2num(k);
        beta = str2num(beta);
    end

    c_infinity = 564;  % salt concentration in mol/m$^3$ at infinity
    b = 10^(-6);  % thickness in meters of absorber
    P = 1400000;  % pressure difference in Pascals
kappaOverMu = 9.72*10^(-20); % kappa/\mu in m^3 s/kg
D = 10^(-9); % diffusion coefficient in m^2/s
R = 8.314; % ideal gas constant in J /K/mol
T = 290; % temperature in Kelvin
Atot = (b*D*k/P/kappaOverMu)^2; % total area of absorbers in m^2
L0 = sqrt(Atot)/\beta; % overall length of graph in m
Ra = sqrt( Atot/2^(G+3)/\pi );
xi = (P-c_{\text{\inf}}*R*T)/P;

50 origin = [0.0 0.0];
B = ones(2^G, 1); % right hand side of equation

global xEnd;
global yEnd;
global endPointCount;
xEnd = zeros(2^(G-1), 1);
yEnd = zeros(2^(G-1), 1);
endPointCount = 0;

60 condCount = 0; % how many times invertibility condition satisfied
noovertCount = 0; % how many times condition of no overlap satisfied
invertCount = 0; % count total number of matrix inversions
errorCount = 0; % number of error percentages of linear approximation

if ( G==2 | G==3 | G==4) 
    maxIteri = 200;
    maxIterj = 200;
else
    maxIteri = 40;
    maxIterj = 67;
end
W = zeros(maxIteri, maxIterj);
sotOverFot = zeros(maxIteri, maxIterj);

for i = 1 : 1 : maxIteri
    for j = 1 : 1 : maxIterj
        switch G
            case 2
\[ r(i) = .5 + .005 \times i; \]
\[ \alpha(j) = .9 + .0015 \times j; \]

**Case 3**
\[ r(i) = .5 + .005 \times i; \]
\[ \alpha(j) = .9 + j \times .0015; \]

**Case 4**
\[ r(i) = .5 + .005 \times i; \]
\[ \alpha(j) = .9 + j \times .0015; \]

**Case 5**
\[ r(i) = .52 + .0015 \times i; \]
\[ \alpha(j) = 1.10 + j \times .0015; \]

**Case 6**
\[ r(i) = .50 + .0015 \times i; \]
\[ \alpha(j) = 1.11 + j \times .0015; \]

**Case 7**
\[ r(i) = .49 + .0015 \times i; \]
\[ \alpha(j) = 1.11 + j \times .0015; \]

**Case 8**
\[ r(i) = .49 + .0015 \times i; \]
\[ \alpha(j) = 1.11 + j \times .0015; \]

**Case 9**
\[ r(i) = .48 + .0015 \times i; \]
\[ \alpha(j) = 1.12 + j \times .0015; \]

**Case 10**
\[ r(i) = .48 + .0015 \times i; \]
\[ \alpha(j) = 1.13 + j \times .0015; \]

```matlab
if ( r(i) == .5 )
    10 = 2*G;
else
    10 = 2*(1 - (2*r(i))^G)/(1 - 2*r(i));
end
L = L0 / 10;
```

\[ \text{stemAngle} = \pi/2; \]
\[ \text{endPointCount} = 0; \]
\[ \text{turtleGraph( origin(1), origin(2), L, stemAngle, 0, G, r(i), alpha(j) );} \]
\[ xEndu = xEnd; \]
\[ yEndu = yEnd; \]
stemAngle = −̂pi/2;
endPointCount = 0;
turtleGraph( origin(1), origin(2), L, stemAngle, 0, G, r(i), alpha(j) );
xEndv = xEnd;
yEndv = yEnd;

X = [xEndu ; xEndv];
Y = [yEndu ; yEndv];

% find W(i,j)
skipW = false;
for ii = 1 : 2^G
    for jj = 1 : 2^G
        if ii == jj
            A(ii,jj) = 1 + k/sqrt(4*pi*2^G);
        else
            if ( norm( [ X(ii) Y(ii) ] − [ X(jj) Y(jj) ] ) < 2 * Ra)
                W(i,j) = 0.0;
                skipW = true;
                break;
            else
                % disp( 'calculated A');
                A(ii,jj) = k*beta/(4*pi*2^G)/norm([X(ii) Y(ii)]−[X(jj) Y(jj)]);
            end
        end
    end
end

detA(i,j) = det(A);
if ( ~skipW )
    % check if sufficient condition for invertibility of A is satisfied
    nooverlapCount = nooverlapCount + 1;
    if ( max(sum(A-eye(2^G)*(1+k/sqrt(4*pi*2^G)),2)) < 1 + k/sqrt(4*pi*2^G))
        condCount = condCount + 1;
    end
end

if ( abs(det(A)) < .00000001 )
```matlab
disp(strcat('not invertible for ratio = ', num2str(r(i)), ' and angle = ', num2str(alpha(j))));
W(i,j) = 0.0;
skipW = true;
else
  % disp('calculated c');
  c = A \ B;
  % keep track of maximum off diagonal entry
  invertCount = invertCount + 1;
  maxOffDiagonal( invertCount ) = max(max(A - eye(2^G) ... 
* (1+k/sqrt(4*pi*2^G))));
end

if (~skipW)
  c Dimensional = c * (c infinity - P/R/T) + P/R/T;
  % debug code to check if nodes producing negative W
  Wij = kappaOverMu*4*pi*2^(-1)*Ra^2/b*(P-c Dimensional*R*T);
  for kk = 1 : size(c Dimensional)
    if ( Wij(kk) < 0 ) % negative water flow from absorber?
      disp(strcat('Wij(kk) = ', num2str(Wij(kk))));
    end
  end % for kk
%}

% calculate errors in linear approximation
for kk = 1 : size(c)
  % first order term = fot
  fot(kk) = (1+k/sqrt(4*pi*2^G))*c(kk);
  sot(kk) = k*x/z/sqrt(4*pi*2^G)*c(kk)^2;
  fot_sum = 0;
  sot_sum = 0;
  for jj = 1 : size(c)
    if ( kk ~= jj )
      fot_sum = fot_sum+c(jj)/norm([X(kk) Y(kk)]-[X(jj) Y(jj)]);
      sot_sum = sot_sum+c(jj)^2/norm([X(kk) Y(kk)]-[X(jj) Y(jj)]);
    end % if kk . . . .
  end % for jj
  fot_sum = k*beta/4/pi/2^G*fot_sum;
  sot_sum = k*beta*x/z/4/pi/2^G*sot_sum;
end
```

\begin{align*}
  \text{fot}(kk) &= \text{fot}(kk) + \text{fot} \cdot \text{sum}; \\
  \text{sot}(kk) &= \text{sot}(kk) + \text{sot} \cdot \text{sum}; \\
\end{align*}

\text{end} \% \text{for} \ kk

\% \text{calculate error in second order term}
\text{sotOverFot}(i,j) = \text{norm} \left( \text{sot} \right) / \text{norm} \left( \text{fot} \right);

\% \text{calculate average size of} \ \tilde{c} 
\text{c_tilde_mean}(i,j) = \text{sum}(c) / \text{size}(c,1);

\text{errorCount} = \text{errorCount} + 1;

W(i,j) = \text{sum}(L0 \ast D \ast k \ast \text{beta} \ast xi \ast c / 2^G);

\text{end} \% \text{not skip} W

\text{end} \% \text{not skip} W

\text{end} \% \text{for} \ j

\text{end} \% \text{for} \ i

\% \text{get optima for ratio} \ (a) \ \text{and angle} \ (b)
[a \ b] = \text{find} \left( W = \text{max} \left( \text{max}(W) \right) \right);

\text{switch} \ G
\begin{align*}
\text{case} \ 2 & \quad \text{optRatio} = .5 + .005 \ast a; \\
& \quad \text{optAngle} = .9 + .0015 \ast b; \\
\text{case} \ 3 & \quad \text{optRatio} = .5 + .005 \ast a; \\
& \quad \text{optAngle} = .9 + b \ast .0015; \\
\text{case} \ 4 & \quad \text{optRatio} = .5 + .005 \ast a; \\
& \quad \text{optAngle} = .9 + b \ast .0015; \\
\text{case} \ 5 & \quad \text{optRatio} = .52 + .0015 \ast a; \\
& \quad \text{optAngle} = 1.10 + b \ast .0015; \\
\text{case} \ 6 & \quad \text{optRatio} = .50 + .0015 \ast a; \\
& \quad \text{optAngle} = 1.11 + b \ast .0015; \\
\text{case} \ 7 & \quad \text{optRatio} = .49 + .0015 \ast a; \\
& \quad \text{optAngle} = 1.11 + b \ast .0015; \\
\text{case} \ 8 & \quad \text{optRatio} = .49 + .0015 \ast a; \\
& \quad \text{optAngle} = 1.11 + b \ast .0015; \\
240 & \quad \text{optRatio} = .49 + .0015 \ast a; \\
& \quad \text{optAngle} = 1.11 + b \ast .0015;
\end{align*}
case 9
    optRatio = .48 + .0015 * a;
    optAngle = 1.12 + b * .0015;
end case 10
    optRatio = .48 + .0015 * a;
    optAngle = 1.13 + b * .0015;
end

fid = fopen('desal10output.txt', 'a');
250 disp(strcat('Generation = ', num2str(G)));
    fprintf(fid, ['r\n', 'Generation = ', num2str(G), 'r\n']);
    disp([r'no overlap condition satisfied ', ...
          num2str(nooverlapCount/maxIteri/maxIterj*100), '% of the time']);
    fprintf(fid, [r'no overlap condition satisfied ', ...
          num2str(nooverlapCount/maxIteri/maxIterj*100), '% of the time', 'r\n']);
    disp([r'invertibility condition satisfied ', ...
          num2str(condCount/maxIteri/maxIterj*100), '% of the time']);
    fprintf(fid, [r'invertibility condition satisfied ', ...
          num2str(condCount/maxIteri/maxIterj*100), '% of the time', 'r\n']);
260 disp([r'diagonal element = ', num2str(l+k)]);
    fprintf(fid, [r'diagonal element = ', num2str(l+k), 'r\n']);
    disp([r'average maximum off diagonal element = ', ...
          num2str(sum(maxOffDiagonal) / invertCount)]);
    fprintf(fid, [r'average maximum off diagonal element = ', ...
          num2str(sum(maxOffDiagonal) / invertCount), 'r\n']);
    disp([r'maximum off diagonal element = ', num2str(max(maxOffDiagonal))]);
    fprintf(fid, [r'maximum off diagonal element = ', num2str(max(maxOffDiagonal)), 'r\n']);
270 disp([r'average error = ', num2str(sum(sum(sotOverFot)) / errorCount)]);
    fprintf(fid, [r'average error = ', num2str(sum(sum(sotOverFot)) / errorCount), 'r\n']);
    disp([r'average c_tilde = ', num2str(sum(sum(c_tilde_mean)) / errorCount)]);
    fprintf(fid, [r'average c_tilde = ', num2str(sum(sum(c_tilde_mean)) / errorCount), 'r\n']);
    disp([r'maximum error = ', num2str(max(max(sotOverFot)))]);
    fprintf(fid, [r'maximum error = ', num2str(max(max(sotOverFot))), 'r\n']);
    disp([r'optimal water output = ', num2str(max(max(W)))]);
    fprintf(fid, [r'optimal water output = ', num2str(max(max(W))), 'r\n']);
280 fprintf(fid, [r'optimal ratio = ', num2str(optRatio), 'r\n']);
disp(['optimal angle =', num2str(optAngle)]);
fprintf(fid, ['optimal angle =', num2str(optAngle), '
']);
fclose(fid);
save strcat('G', num2str(G), 'BCConstPress'), 'W', 'r', 'alpha', 'detA', ...
'sotOverFot');

return

fid = fopen('desal10output.txt', 'a');
runTime = datenum(now);
fprintf(fid, [runTime, '
']);
fclose(fid);

disp('------------------------ Generation = 2 ------------------------');

disp(' k = 2.15, beta = .0158 ')
desal10 2 2.15 .0158;

disp('------------------------ Generation = 3 ------------------------');

disp(' k = 2.15, beta = .0158 ')
desal10 3 2.15 .0158;

disp('------------------------ Generation = 4 ------------------------');

disp(' k = 2.15, beta = .0158 ')
desal10 4 2.15 .0158;

disp('------------------------ Generation = 5 ------------------------');

disp(' k = 2.15, beta = .0158 ')
desal10 5 2.15 .0158;

disp('------------------------ Generation = 6 ------------------------');

disp(' k = 2.15, beta = .0158 ')
desal10 6 2.15 .0158;

disp('------------------------ Generation = 7 ------------------------');
disp(' k = 2.15, beta = 0.158 ')
desal10 7 2.15 .0158;
disp('Generation = 8 ');
desal10 8 2.15 .0158;
disp('Generation = 9 ');
desal10 9 2.15 .0158;
disp('Generation = 10 ');
desal10 10 2.15 .0158;

function doPlots( extension )

% file name = doPlots.m
%
% creates all plots for desalination paper
% usage:
% doPlots extension
% where extension = jpg (makes titles) or eps (no titles, saved in eps format)
%
if (nargin ~= 1)
    help doPlots;
    return;
end

doFittedPlots4 extension
doTurtleGraphs extension
doContours extension
function doFittedPlots( extension )
%
% file name = doFittedPlots.m
%
% creates fitted plots for optimal ratio, angle, and water production rates
% versus generation G
% usage:
%   doFittedPlots extension
%
% where extension = jpg (makes titles) or eps (no titles, saved in eps format)
%
if (nargin ~= 1)
    help doFittedPlots;
    return;
end

x = [2 3 4 5 6 7 8 9 10];
xc = 2:.02:10;
y1 = [1.19 .68 .57 .54 .53 .52 .51 .51];
y2 = [1.05 1.08 1.14 1.17 1.17 1.17 1.17 1.17 1.2];
y3 = 10^-13*[7.45 7.95 8.32 8.57 8.71 8.76 8.74 8.67 8.56];

f1 = .5 + 3.8168 * xc.^( -2.6225 );
f2 = 1.20 -.151 * exp( -.231 * xc );
f3 = 5.741 * 10^-13 * exp( .7119 * xc );

h1 = figure(1);
h2 = figure(2);
h3 = figure(3);

figure(1)
plot(xc,f1, 'LineWidth', 2);
hold on
plot(x,y1,'--rs', 'LineWidth', 2,...
     'MarkerEdgeColor','k' ,...
     'MarkerFaceColor','g' ,...
     'MarkerSize',10);
set(gca,'FontSize',22);
set(gca,'LineWidth',2);
xlabel('G');
ylabel('r');
if(strcmp(extension,'jpg'))
title('Optimal Ratio');
saveas(h1,'OptimalRatio','jpg');
elseif(strcmp(extension,'eps'))
saveas(h1,'OptimalRatio.eps','epsc');
else
    help doFittedPlots;
    return;
end

figure(2)
plot(xc,f2,'LineWidth',2);
hold on
plot(x,y2,'--rs','LineWidth',2,...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor','g',...
    'MarkerSize',10);
set(gca,'FontSize',22);
set(gca,'LineWidth',2);
xlabel('G');
ylabel('$\alpha$ in radians');
if(strcmp(extension,'jpg'))
title('Optimal Angle');
saveas(h2,'OptimalAngle','jpg');
elseif(strcmp(extension,'eps'))
saveas(h2,'OptimalAngle.eps','epsc');
else
    help doFittedPlots;
    return;
end

figure(3)
plot(xc,f3,'LineWidth',2);
hold on
plot(x,y3,'--rs','LineWidth',2,...
    'MarkerEdgeColor','k',...
set(gca,'FontSize',22);
set(gca,'LineWidth',2)
xlabel('G');
ylabel('W in m$^3/$s');
if(strcmp(extension,'.jpg'))
title('Optimal Water Production Rate');
saveas(h3,'OptimalWaterProductionRate','jpg');
elseif(strcmp(extension,'.eps'))
saveas(h3,'OptimalWaterProductionRate.epsc','epsc');
else
help doFittedPlots;
return;
end

pause
delete(h1)
delete(h2)
delete(h3)

function doTurtleGraphs( extension )
%
% file name = doTurtleGraphs.m
%
% creates biturtle graph plots for the optimal ratios and angles
% usage:
% doTurtleGraphs extension
% where extension = jpg (makes titles) or eps (no titles, saved in
% eps format)
%
if (nargin ~= 1)
help doTurtleGraphs;
return;
end

h1 = figure(1);
h2 = figure(2);
h3 = figure(3);

figure(1)
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
dtg258;

if (strcmp(extension, 'jpg'))
    title('Optimal Turtle Graphs for Generations 2, 5, and 8');
    saveas(h1, 'OptimalTurtleGraphsforGenerations258', 'jpg');
elseif (strcmp(extension, 'eps'))
    title('Generations 2, 5, and 8');
    saveas(h1, 'OptimalTurtleGraphsforGenerations258.eps', 'epsc');
else
    help doTurtleGraphs;
    return;
end

figure(2)
dtg369;
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
if (strcmp(extension, 'jpg'))
    title('Optimal Turtle Graphs for Generations 3, 6, and 9');
    saveas(h2, 'OptimalTurtleGraphsforGenerations369', 'jpg');
elseif (strcmp(extension, 'eps'))
    title('Generations 3, 6, and 9');
    saveas(h2, 'OptimalTurtleGraphsforGenerations369.eps', 'epsc');
else
    help doTurtleGraphs;
    return;
end

figure(3)
dtg4710;
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
if (strcmp(extension, 'jpg'))
    title('Optimal Turtle Graphs for Generations 4, 7, and 10');

88
saveas(h3, 'OptimalTurtleGraphsforGenerations4710', 'jpg');

elseif (strcmp(extension, 'eps'))
    title('Generations 4, 7, and 10');
    saveas(h3, 'OptimalTurtleGraphsforGenerations4710.eps', 'epsc');
else
    help doTurtleGraphs;
    return;
end

figure(4)
dtg7;
set(gca,'FontSize',22);
set(gca,'LineWidth',2)
if (strcmp(extension, 'jpg'))
    title('Optimal Turtle Graph, G=7, r=.52, alpha=1.17');
    saveas(h4, 'OptimalTurtleGraphforGeneration7', 'jpg');
elseif (strcmp(extension, 'eps'))
    saveas(h4, 'OptimalTurtleGraphforGeneration7.eps', 'epsc');
else
    help doTurtleGraphs;
    return;
end

pause
delete(h1)
delete(h2)
delete(h3)
delete(h4)

% file name dtg258.m
% draws turtle graphs

drawTurtleGraph(0,0,.14705, pi/2, 0, 2, 1.2, 1.047, 4, 'r')
hold on
drawTurtleGraph(0,0,.083877, pi/2, 0, 2, 1.2, 1.047, 4, 'r')
drawTurtleGraph(0,0,.083877, -pi/2, 0, 2, 0.5, 544, 1.151, 2, 'g')
drawTurtleGraph(0,0,.083877, -pi/2, 0, 2, 0.5, 544, 1.151, 2, 'g')
drawTurtleGraph(0,0,.056027, pi/2, 0, 2, 0.8, 5155, 1.167, 1, 'b')
drawTurtleGraph(0,0,.056027, -pi/2, 0, 2, 0.8, 5155, 1.167, 1, 'b')
function drawTurtleGraph (xStart, yStart, len, angle, currGen, maxGen, ...
590  ratio, angleIncrement, width, color)

% recursive function to draw turtleGraph
% (xStart, yStart): recursed initial starting point of turtle graph
% len: recursed power of ratio times normalized turtle length
% angle: recursed current angle
% currGen: recursed current turtle graph generation
% maxGen: recursion ends when maxGeneration reached
% angleIncrement: amount in radians to change the angle on each recursion
% ratio: factor by which the current value of len is multiplied to

% produce some power of ratio
% width: widths of line to be plotted
% color: colors of line to be plotted
%
xNew = xStart + len*cos(angle);            % calculate new x coordinate
yNew = yStart + len*sin(angle);            % calculate new y coordinate
currGen = currGen + 1;                    % increment recursion count
%
% draw a line
line([xNew; xStart], [yNew; yStart], 'LineWidth', width, 'Color', color);

if currGen < maxGen               % recursion not done?
    % fork a new pair of turtles
drawTurtleGraph(xNew, yNew, len*ratio, angle + angleIncrement, currGen,...
    maxGen, ratio, angleIncrement, width, color);
drawTurtleGraph(xNew, yNew, len*ratio, angle - angleIncrement, currGen,...
    maxGen, ratio, angleIncrement, width, color);
else % recursion done
    return
end

function doContours( extension )
%
% file name = doContours.m
%
% creates contour plots for generations 3, 5, and 7
% usage:
%      doContours extension
%
% where extension = jpg (makes titles) or eps (no titles, saved in
%      eps format)
%
if (nargin ~= 1)
    help doContours;
    return;
end

h1 = figure(1);
h2 = figure(2);
h3 = figure(3);

figure(3)
ctrs 7
if (strcmp(extension, 'jpg'))
    title('Water Production Contours for Generation 7');
saveas(h3, 'WaterProductionContoursforGeneration7', 'jpg');
elseif (strcmp(extension, 'eps'))
    title('Generation 7');
saveas(h3, 'WaterProductionContoursforGeneration7.eps', 'epsc');
else
    help doContours;
    return;
end

figure(1)
ctrs 3
if (strcmp(extension, 'jpg'))
    title('Water Production Contours for Generation 3');
saveas(h1, 'WaterProductionContoursforGeneration3', 'jpg');
elseif (strcmp(extension, 'eps'))
    title('Generation 3');
saveas(h1, 'WaterProductionContoursforGeneration3.eps', 'epsc');
else
    help doContours;
    return;
end

figure(2)
ctrs 5
if (strcmp(extension, 'jpg'))
    title('Water Production Contours for Generation 5');
saveas(h2, 'WaterProductionContoursforGeneration5', 'jpg');
elseif (strcmp(extension, 'eps'))
    title('Generation 5');
saveas(h2, 'WaterProductionContoursforGeneration5.eps', 'epsc');
else
    help doContours;
    return;
function ctrs(G)

% script to plot contour for generation G
% usage: ctrs G
% G: generations of turtle graph
%

if (nargin ~= 1)
    help ctrs;
    return;
end

load(strcat('G',G,' BCConstPress.mat'));
for i = 1:199; for j = 1:104; if (W(i,j) == 0.0); W(i,j) = 1.0; end; end; end;
    minW = min(min(W));
for i = 1:199; for j = 1:104; if (W(i,j) == 1.0); W(i,j) = minW; end; end; end;
contour(W,100);
set(gca,'FontSize',22);
set(gca,'LineWidth',2);
xlabel('\alpha in radians');
ylabel('r');
set(gca,'XtickLabel',{'.6';'.12';'.18';'.24';'.30'})
set(gca,'YtickLabel',{'.5';'.10';'.15'})
hold on

% file name desalG3VaryRatio.m
% script with no arguments to find water production versus two independent
% ratios for generation 2 binary graph with fixed length
clear all  % just in case

P = 1400000;
c_infinity = 564;  % salt concentration of seawater in mol/m^3 at infinity
Ra = sqrt(.00025/32/pi);  % node radius in meters
L0 = 1.0;  % fixed overall length in meters of graph
b = 10^(-6);  % thickness in meters of absorber
kappaOverMu = 9.72*10^(-20);  % form of permeability, kappa/mu in m^3 s/kg

D = 10^(-9);  % diffusion coefficient in m^2/s
R = 8.314;  % ideal gas constant in J/K/mol
T = 290;  % temperature in Kelvin
Aa = .00025;
k = sqrt(Aa)*kappaOverMu*P/b/D;  % dimensionless parameter in equations
beta = sqrt(Aa)/L0;

xi = (P-c_infinity*R*T)/P;

origin = [0.0 0.0];

G = 3;
B = ones(2^G, 1);  % right hand side of equation

angle = 1.10;  % use optimal angle found in earlier studies
ratio = .67;
l0 = 2 * ( 1 + 2 * ratio + 4 * ratio^2 );  % unscaled length
L = L0 / 10;  % length scale factor

ratios = [.172 1.172; .272 1.072; .372 .972; .472 .872; .572 .772; .672 ... .672; .772 .572; .872 .472; .972 .372; 1.072 .272 ; 1.172 .172];

for i = 1 : 11
    ratioL = ratios(i,1);
ratioR = ratios(i,2);

    stemAngle = pi/2;
    u0 = origin + L * [cos( stemAngle) sin( stemAngle) ];
    u0L = u0 + ratioL * L*[cos(stemAngle+angle) sin(stemAngle+angle)];
    u0R = u0 + ratioR * L*[cos(stemAngle-angle) sin(stemAngle-angle)];
    u0LL = u0L + ratioL^2 * L*[cos(stemAngle+2*angle) sin(stemAngle+2*angle)];
\[ u_{0LR} = u_0L + \text{ratio}_L \cdot \text{ratio}_R \cdot L \cdot [\cos(\text{stemAngle}) \sin(\text{stemAngle})]; \]
\[ u_{0RL} = u_0R + \text{ratio}_R \cdot \text{ratio}_L \cdot L \cdot [\cos(\text{stemAngle}) \sin(\text{stemAngle})]; \]
\[ u_{0RR} = u_0R + \text{ratio}_R \cdot L \cdot \left[ \cos(\text{stemAngle} - 2\cdot\text{angle}) \sin(\text{stemAngle} - 2\cdot\text{angle}) \right]; \]
\[ \text{stemAngle} = -\pi/2; \]
\[ v_0 = \text{origin} + L \cdot \left[ \cos(\text{stemAngle}) \sin(\text{stemAngle}) \right]; \]
\[ v_{0L} = v_0 + \text{ratio}_L \cdot L \cdot \left[ \cos(\text{stemAngle} + \text{angle}) \sin(\text{stemAngle} + \text{angle}) \right]; \]
\[ v_{0R} = v_0 + \text{ratio}_R \cdot L \cdot \left[ \cos(\text{stemAngle} - \text{angle}) \sin(\text{stemAngle} - \text{angle}) \right]; \]
\[ v_{0LL} = v_{0L} + \text{ratio}_L \cdot L \cdot \left[ \cos(\text{stemAngle} + 2\cdot\text{angle}) \sin(\text{stemAngle} + 2\cdot\text{angle}) \right]; \]
\[ v_{0LR} = v_{0L} + \text{ratio}_L \cdot \text{ratio}_R \cdot L \cdot \left[ \cos(\text{stemAngle}) \sin(\text{stemAngle}) \right]; \]
\[ v_{0RL} = v_{0R} + \text{ratio}_R \cdot \text{ratio}_L \cdot L \cdot \left[ \cos(\text{stemAngle} + 2\cdot\text{angle}) \sin(\text{stemAngle} + 2\cdot\text{angle}) \right]; \]
\[ v_{0RR} = v_{0R} + \text{ratio}_R \cdot L \cdot \left[ \cos(\text{stemAngle} - 2\cdot\text{angle}) \sin(\text{stemAngle} - 2\cdot\text{angle}) \right]; \]

\textbf{figure}(i);
\textbf{XY} = [\text{origin} \quad u_0];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{hold on}
\textbf{XY} = [ u_0 \quad u_{0L}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ u_0 \quad u_{0R}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ u_{0L} \quad u_{0LL}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ u_{0L} \quad u_{0LR}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ u_{0R} \quad u_{0RL}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ u_{0R} \quad u_{0RR}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ \text{origin} \quad v_0];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ v_0 \quad v_{0L}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ v_0 \quad v_{0R}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ v_{0L} \quad v_{0LL}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ v_{0L} \quad v_{0LR}];
\textbf{line}(\text{XY(:,1), XY(:,2)} , 'LineWidth', 3);
\textbf{XY} = [ v_{0R} \quad v_{0RL}];
line( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ v0R ; v0RR];
line( XY(:,1), XY(:,2), 'LineWidth', 3);
set(gca, 'FontSize',22);
set(gca,'LineWidth',2)
axis equal
pause

XY = [u0LL ; u0LR ; u0RL ; u0RR ; v0LL ; v0LR ; v0RL ; v0RR];

810
% find W(i)
skipW = false;
for ii = 1 : 2^G
    for jj = 1 : 2^G
        if ii == jj
            A(ii,jj) = 1 + k/sqrt(4*pi*2^G);
        else
            if ( norm([XY(ii,1) XY(ii,2)] - [XY(jj,1) XY(jj,2)]) < 2*Ra)
                W(i ) = 0.0;
                skipW = true;
                break;
            else
                A(ii,jj) = k*beta/(4*pi*2^G)/norm([XY(ii,1) XY(ii,2)]... -[XY(jj,1) XY(jj,2)]);
            end
        end % if ii == jj
    end % for jj
if ( skipW )
    break;
end % for ii

830

if (~skipW && (abs(det(A)) < .00000001) )
    disp(strcat('not invertible for ratioL = ', num2str(ratioL), ...
    ' and ratioR = ', num2str(ratioR)));
    W(i ) = 0.0;
    skipW = true;
else
    c = A \ B;
end

96
if (~skipW)
    W(i) = sum(L0*D*k*beta*x(i)*c/2*G);
end

end % for i

W

h=figure(12);
850 plot([1 2 3 4 5 6 7 8 9 10 11], W, 'rs', 'LineWidth',2,...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor','g',...
    'MarkerSize',10);
hold on
set(gca, 'FontSize',22);
set(gca, 'LineWidth',2)
xlabel('rL-rR')
ylabel('W in m^3/s');
xlim([0 12]);
860 set(gca, 'XTick', [1 6 11 ]);
set(gca, 'XTickLabel', [-1.0 0.0 1.0] );
saveas(h,'OptimalWaterProductionG3VersusRatio.epsl','epsc')
return

% file name desalG3VaryAngle.m
% script with no arguments to find water production versus two independent
% angle for generation 2 binary graph with fixed length

870 clear all % just in case

P = 1400000;
c_infinity = 564; % salt concentration of seawater in mol/m^3 at infinity
Ra = sqrt(0.00525/32/pi); % node radius in meter
L0 = 1.0; % fixed overall length in meters of graph
b = 10^(-6); % thickness in meters of absorber
kappaOverMu = 9.72*10^(-20); % form of permeability, kappa/mu in m^3 s/kg

D = 10^(-9); % diffusion coefficient in m^2/s
R = 8.314; % ideal gas constant in J/K/mol
\( T = 290; \quad \% \) temperature in Kelvin

\( Aa = 0.00025; \)

\( k = \sqrt{Aa} \ast \frac{\kappa}{\mu} \ast \frac{P}{b/D}; \quad \% \) dimensionless parameter in equations

\( \beta = \sqrt{Aa} / L0; \)

\( \xi = (P - c_{\infty} \ast R \ast T) / P; \)

\( \text{origin} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}; \)

\( G = 3; \)

\( B = \text{ones}(2^G, 1); \quad \% \) right hand side of equation

\( \angle = 1.10; \quad \% \) use optimal angle found in earlier studies

\( \text{ratio} = 0.67; \)

\( L0 = 2 \ast (1 + 2 \ast \text{ratio} + 4 \ast \text{ratio}^2); \quad \% \) unscaled length

\( \text{L} = L0 / 10; \quad \% \) length scale factor

\( \text{for} \ i = -5 : 5 \)

\( \text{delta} = i \ast \angle / 5; \)

\( \angle L = \angle + \text{delta}; \)

\( \angle R = \angle - \text{delta}; \)

\( \text{stemAngle} = \pi / 2; \)

\( \text{u0} = \text{origin} + \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}) \sin(\text{stemAngle}) \end{bmatrix}; \)

\( \text{u0L} = \text{u0} + \text{ratio} \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+\angle L) \sin(\text{stemAngle}+\angle L) \end{bmatrix}; \)

\( \text{u0R} = \text{u0} + \text{ratio} \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}-%angle R) \sin(\text{stemAngle}-%angle R) \end{bmatrix}; \)

\( \text{u0LL} = \text{u0L} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+2\ast\angle L) \sin(\text{stemAngle}+2\ast\angle L) \end{bmatrix}; \)

\( \text{u0LR} = \text{u0L} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+\angle L-%angle R) \sin(\text{stemAngle}+\angle L-%angle R) \end{bmatrix}; \)

\( \text{v0} = \text{origin} + \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}) \sin(\text{stemAngle}) \end{bmatrix}; \)

\( \text{v0L} = \text{v0} + \text{ratio} \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+\angle L) \sin(\text{stemAngle}+\angle L) \end{bmatrix}; \)

\( \text{v0R} = \text{v0} + \text{ratio} \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}-%angle R) \sin(\text{stemAngle}-%angle R) \end{bmatrix}; \)

\( \text{v0LL} = \text{v0L} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+2\ast\angle L) \sin(\text{stemAngle}+2\ast\angle L) \end{bmatrix}; \)

\( \text{v0LR} = \text{v0L} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}+\angle L-%angle R) \sin(\text{stemAngle}+\angle L-%angle R) \end{bmatrix}; \)

\( \text{v0RL} = \text{v0R} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}-%angle R+\angle L) \sin(\text{stemAngle}-%angle R+\angle L) \end{bmatrix}; \)

\( \text{v0RR} = \text{v0R} + \text{ratio}^2 \ast \text{L} \ast \begin{bmatrix} \cos(\text{stemAngle}-%2\ast\angle R) \sin(\text{stemAngle}-%2\ast\angle R) \end{bmatrix}; \)
\* L*[{\cos(\text{stemAngle}-\text{angleR}+\text{angleL}) \sin(\text{stemAngle}-\text{angleR}+\text{angleL})}];
\v_{0RR} = \v_{0R} + \text{ratio}^2 \* L*[{\cos(\text{stemAngle}-2\times\text{angleR}) \sin(\text{stemAngle}-2\times\text{angleR})}];

\begin{verbatim}
figure( i + 6);
XY = [ origin ; \u_0 ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
hold on
XY = [ \u_0 ; \u_0L ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
end
XY = [ \u_0L ; \u_0LR ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \u_0L ; \u_0LL ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \u_0R ; \u_0RL ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \u_0R ; \u_0RR ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ origin ; \v_0 ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \v_0 ; \v_0L ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \v_0L ; \v_0LR ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \v_0L ; \v_0LL ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \v_0R ; \v_0RL ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
XY = [ \v_0R ; \v_0RR ];
\text{line}( XY(:,1), XY(:,2), 'LineWidth', 3);
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
axis equal
pause
XY = [ \u_0LL ; \u_0LR ; \u_0RL ; \u_0RR ; \v_0LL ; \v_0LR ; \v_0RR ];
\end{verbatim}

99
% find W(i)

skipW = false;
for ii = 1 : 2^G
  for jj = 1 : 2^G
    if ii == jj
      A(ii, jj) = 1 + k/sqrt(4*pi*2^G);
    else
      if (norm([XY(ii, 1) XY(ii, 2)] - [XY(jj, 1) XY(jj, 2)]) < 2*Ra)
        W(i + 6) = 0.0;
      else
        A(ii, jj) = k*beta/(4*pi*2^G) . . . / norm([XY(ii, 1) XY(ii, 2)] - [XY(jj, 1) XY(jj, 2)]);
      end
    end
  end
  if (skipW)
    break;
  end
end

if (~skipW & (abs(det(A)) < .00000001))
  disp(strcat('not invertible for ratioL = ', ...
              num2str(ratioL), ' and ratioR = ', num2str(ratioR)));
  W(i + 6) = 0.0;
  skipW = true;
else
  c = A \\ B;
end

if (~skipW)
  W(i + 6) = sum(L0*D*k*beta*xi*c/2^G);
end

end % for i

W

h=figure(12)
plot([1 2 3 4 5 6 7 8 9 10 11], W, 'rs', 'LineWidth', 2, ...
'MarkerEdgeColor', 'k', ...
'MarkerFaceColor', 'g', ...
'MarkerSize', 10);

hold on
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
xlabel('\alpha_L-\alpha_R in radians');
ylabel('W in m$^3$/s');
xlim([0 12]);

set(gca, 'XTick', [1 6 11]);
set(gca, 'XTickLabel', [−2.19 0.0 2.19]);
saveas(h,’OptimalWaterProductionG3VersusAngle.eps’,’epsc’)

return

G=[2 3 4 5 6 7 8 9 10];
W=[7.47 8.01 8.43 8.74 8.97 9.12 9.22 9.27 9.29;
    7.46 7.98 8.38 8.67 8.86 8.97 9.02 9.01 8.97;
    7.45 7.95 8.32 8.57 8.71 8.76 8.74 8.67 8.56;
    7.43 7.91 8.25 8.44 8.52 8.51 8.42 8.27 8.05;
    7.42 7.87 8.16 8.30 8.31 8.22 8.05 7.83 7.42;]
    ]*10^(-13);

G2=2:.1:10;
for i = 1 : 5
    Wn=W(i,:);
    p=polyfit(G,Wn,2);
    W2(i,:) = polyval(p,G2);

    c2 = - p(1);
    Gmax = p(2)/2/c2;
    c1 = p(3) + c2*Gmax^2;
    disp(strcat('for \beta=' , num2str((i-3)/4+1),',\beta_0=', num2str(cl),',\beta_2=',
        num2str(c2),',G_{max}=',num2str(Gmax),'));
end

h = figure(1);
plot(G2, W2(1,:), 'r-', G2, W2(2,:), 'b-', G2, W2(3,:), 'g-', G2, W2(4,:),...
hold on

plot(G, W(1,:), 'rs', G, W(2,:), 'bd', G, W(3,:), 'go', G, W(4,:), 'cp', G, W(5,:), 'mh', 'LineWidth', 2, ...
     'MarkerEdgeColor', 'k', ...
     'MarkerFaceColor', 'g', ...
     'MarkerSize', 10);

legend('\beta=.5\beta_0', '\beta=.75\beta_0', '\beta=\beta_0', ...
     '\beta=1.25\beta_0', '\beta=1.5\beta_0', 'Location', 'BestOutside')

set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
xlabel('G');
ylabel('W in m^3/s');
saveas(h, 'Fig.6.epss', 'eps');

G=[2 3 4 5 6 7 8 9 10];
W=[ 2.11 2.18 2.22 2.25 2.24 2.22 2.19 2.16;
     4.45 4.68 4.84 4.94 4.98 4.96 4.91 4.84;
     7.45 7.95 8.32 8.57 8.71 8.76 8.74 8.67 8.56;
     11.0 11.9 12.6 13.1 13.4 13.5 13.5 13.3;
     15.0 16.5 17.6 18.4 19.0 19.2 19.3 19.3 19.1;]*10^(-13);

G2=2:.1:10;
for i = 1 : 5
    Wn=W(i,:);
    p=polyfit(G,Wn,2);
    W2(i,:)=polyval(p,G2);
    % p = p(1)*x^2 + p(2)*x + p(3)
    c2 = - p(1);
    Gmax = p(2)/2/c2;
    c1 = p(3) + c2*Gmax^2;
    disp(strcat('for k=',num2str((i-3)/4+1),', k_0=', num2str(c1), ', c1=', num2str(c2), ', c2=', num2str(Gmax)));
end
```matlab
h = figure(1);
plot(G2, W2(1,:), '−r', G2, W2(2,:), '−.b', G2, W2(3,:), '−−g', G2, W2(4,:), '.c', G2, W2(5,:), '−.m', 'MarkerSize', 5, 'LineWidth', 2)
hold on
plot(G, W(1,:), 'rs', G, W(2,:), 'bd', G, W(3,:), 'go', G, W(4,:), 'cp', G, W(5,:), 'mh', 'LineWidth', 2, 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g', 'MarkerSize', 10);
legend('k=.5k_0', 'k=.75k_0', 'k=k_0', 'k=1.25k_0', 'k=1.5k_0', ... 'Location', 'BestOutside')
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2)
xlabel('Г');
ylabel('W in m^3/s')
saveas(h, 'Fig.1.7.eps', 'epscl');

y1 = [8.95 8.21 7.44 6.68 5.94];
y2 = [6.25 7.0 7.44 7.76 8.05];
beta0 = .0158;
k0 = 2.15;
x1 = [.5*beta0 .75*beta0 beta0 1.25*beta0 1.5*beta0];
x2 = [.5*k0 .75*k0 k0 1.25*k0 1.5*k0];

h1 = figure(1);
h2 = figure(2);

figure(1);
plot(x1,y1, 'rs', 'LineWidth', 2, ...
     'MarkerEdgeColor', 'k', ...
     'MarkerFaceColor', 'g', ...
     'MarkerSize', 10);
set(gca, 'FontSize', 22);
set(gca, 'LineWidth', 2);
xlabel('eta');
ylabel('G_{max}');
```

103
saveas(h1,'Fig_18.epsc');

figure(2);
plot(x2,y2,'rs','LineWidth',2,...
     'MarkerEdgeColor','k',...
     'MarkerFaceColor','g',...
     'MarkerSize',10);
set(gca,'FontSize',22);
set(gca,'LineWidth',2)
xlabel('k');
ylabel('G_{max}');
saveas(h2,'Fig_19.epsc','epsc');
Appendix G

Code for Perceptron Study

function sp (numInputs, numTrials, showPatternClasses)

%%%%%%%%% input lines to use, type 'sp numInputs numTrials' %
%%%%%%%%% showPatternClass', where numInputs is number of input lines; %
%%%%%%%%% numTrials gives the number of times to train the pattern; %
%%%%%%%%% showPatternClasses is an optional switch to display the pattern %
%%%%%%%%% classes ordered by decreasing magnitude of weight response %

if (nargin == 0)
    help sp;
    break;
elseif (nargin == 1)
    numTrials = 10;
else
    numTrials = str2num(numTrials);
end

numInputs = str2num(numInputs);

if (nargin == 3)
    showPatternClasses = str2num(showPatternClasses);
end

% for larger n, sampleSize will be a very small fraction of total
% population of patterns
largePop = 4;
if (numInputs < largePop)
    sampleSize = 2^(2*numInputs); % the total population of patterns
else
    sampleSize = 10^n(numInputs);
false = 0;
true = 1;

rules=ones(2^numInputs , numInputs+2);
maxTries = 500;
adaptation = .15;  \% constant of proportionality to change weights by
results : column 1 = pattern of input, column 2 = average (over
\% trials) number of time steps to converge, column 3 = average of
\% volume estimates of attractors , column 4 = ave. Hamming distance
\% between inputs in same class, col.s 5 to 4+2^numInputs = decimal
\% value of inputs in order of decreasing response by perceptron
\% (thus the first few include resonant inputs)
results = zeros(sampleSize,4+2^numInputs);
% store steps to converge for each trial
corvergenceResults = zeros(1,sampleSize * numTrials);
results (:,1) = -1;
if (numInputs == 2)
    \% contains the trajectory history for all trials
    history = zeros(2^(2^numInputs), numTrials,...
    maxTries*2^numInputs, numInputs);
end
if (numInputs == 2 | numInputs == 3)
    \% contains converged weight vector for each trial
    attractors = zeros(2^(2^numInputs), numTrials+1, numInputs+2);
end
\% build rules matrix
for m = 1 : 2^numInputs
    \% get binary string
    pattern = dec2bin(m-1,numInputs);
    rules(m,1) = 1; \% first neuron set to one to pick up bias
    for n = 2 : numInputs + 1
        \% convert to decimal arrays of 0's and 1's
        rules (m, n) = bin2dec(pattern(n-1));
    end
end

numConverged = 0;
for s = 1 : 2
    bias = (-1)^s;
for m = 1 : sampleSize  
% attempt to train on output pattern
% prepare the pattern responses
if (numInputs >= largePop)
sample = floor(2^(2*numInputs)*rand(1));
    while (ismember(sample, results(:,1)))
        % don’t retrain on same output pattern
        sample = floor(2^(2*numInputs)*rand(1));
    end
else
    sample = m-1;
end

% string of 0’s and 1’s of length 2^numInputs
responses = dec2bin(sample, 2^numInputs);
for n = 1:2^numInputs
    % the set of 2^numInputs’ responses determines
    % the rule
    rules(n, numInputs+2) = bin2dec(responses(n));
end
volAttractors = 0;  
% rough estimate of volume of attractor
totalChanges = 0;
tryCount = 0;
rawOutputs = zeros(1, 2^numInputs);
for k = 1 : numTrials
    trialChanges = 0;
    % row matrix of weights
    w(2:numInputs+1) = randGrid(10, numInputs);
    w(1) = bias;  % bias fixed at -1
    if (numInputs == 2 | numInputs == 3)
        % counter variable to count the number of weight changes
        historyCount = 1;
        % save copy of current weights
        history(numConverged + 1, k, historyCount, :) ...
            = w(2:numInputs+1);
        % count how many entries in this vector for plotting
        attractors(numConverged + 1, k, numInputs+2) = 1;
    end
    tryCount = 0;  
    % no. of weight changes each 2^numInputs inputs
    while (tryCount < maxTries)  
        % train the weights
        numChanges = 0;
        for n = 1:2^numInputs
if \((w^\top\text{rules}(n,1:\text{numInputs}+1) \leq 0) \ldots\)
& \((\text{rules}(n,\text{numInputs}+2) = 1)\)
\(w(2:\text{numInputs}+1) = w(2:\text{numInputs}+1) + \text{adaptation} \ldots\)
* \text{rules}(n,2:\text{numInputs}+1); 
\text{numChanges} = \text{numChanges} + 1; 
\text{trialChanges} = \text{trialChanges} + 1;
if \((\text{numInputs} = 2)\)
\text{historyCount} = \text{historyCount} + 1; 
\% \text{save copy of current weights}
\text{history}(\text{numConverged} + 1,k,\text{historyCount},1:\text{numInputs}) \ldots 
= w(2:\text{numInputs}+1);
end

elseif \((w^\top\text{rules}(n,1:\text{numInputs}+1) > 0) \& (\text{rules}(n,\text{numInputs}+2) = 0)\)
\(w(2:\text{numInputs}+1) = w(2:\text{numInputs}+1) - \text{adaptation} \ldots\)
* \text{rules}(n,2:\text{numInputs}+1); 
\text{numChanges} = \text{numChanges} + 1; 
\text{trialChanges} = \text{trialChanges} + 1;
if \((\text{numInputs} = 2)\)
\text{historyCount} = \text{historyCount} + 1; 
\% \text{save copy of current weights}
\text{history}(\text{numConverged} + 1,k,\text{historyCount},1:\text{numInputs}) \ldots 
= w(2:\text{numInputs}+1);
end
end
end \% \text{end for loop to check all weight responses}
if \((\text{numChanges} = 0) \% \text{convergence}\)
break \% \text{process the results}
else
\text{totalChanges} = \text{totalChanges} + \text{numChanges};
\text{tryCount} = \text{tryCount} + 1;
end
end \% \text{end while loop to train weights for trial}
if \((\text{tryCount} = \text{maxTries}) \% \text{did not converge, skip}\)
break; \% \text{break out of trial loop and try new random pattern of outputs}
else \% \text{there was convergence, process this trial's results}
\text{convergenceResults}(\text{numConverged} \ast \text{numTrials} + k) = \text{trialChanges};
if \((\text{numInputs} = 2 \mid \text{numInputs} = 3)\)
\text{attractors}(\text{numConverged} +1,k,1:\text{numInputs}) = w(2:\text{numInputs}+1); 
\% \text{identify weights by rule = sample}
attractors(numConverged + 1, 1, numInputs + 1) = sample;
attractors(numConverged + 1, k, numInputs + 2) = historyCount;
end

for i = 1 : 2^numInputs
    rawOutputs(i) = rawOutputs(i) + w*rules(i,1:numInputs+1)’;
end

volAttractor = 0;
for i = 2 : numInputs + 1
    volAttractor = volAttractor + w(i)^2;
end

% rough estimate of volume of attractor
volAttractor = sqrt(volAttractor)^numInputs;
volAttractors = volAttractors + volAttractor;

end % end the per trial processing of results
end % end loop on trial number

if (tryCount < maxTries)
    numConverged = numConverged + 1;
    results(numConverged,1) = sample;
    results(numConverged,2) = totalChanges / numTrials;
    results(numConverged,3) = volAttractors / numTrials;
    attractors(numConverged, numTrials+1, 1) = sample; % outputs
    % number of steps
    attractors(numConverged, numTrials+1, 2) = totalChanges / numTrials;
    % in results(numConverged,4) store average Hamming distance between % inputs of same class
    count = 0;
    results(numConverged,4) = 0;
    for i = 1 : 2^numInputs-1
        for j = i+1 : 2^numInputs
            if (rules(i,numInputs+2) == rules(j,numInputs+2))
                results(numConverged,4) = results(numConverged,4) + ...
                hammingDistance(rules(i,2:numInputs+1), rules(j,2:numInputs+1));
                count = count + 1;
            end
        end
    end
    results(numConverged,4) = results(numConverged,4) / count;
    % in results(numConverged,5:2^numInputs+4) store decimal values
    % of input patterns in order of decreasing response
    if ((nargin == 3)&(showPatternClasses==true))
tmp = zeros (2^numInputs, 2);
for i = 1 : 2^numInputs
    tmp(i,1) = binArray2Dec(rules(i,2:numInputs+1));
    % study the raw output of the perceptron
    tmp(i,2) = rawOutputs(i)/numTrials;
end
tmp = sortrows(tmp,2);
results(numConverged,5:2*numInputs+4) = tmp(:,1)';
results(numConverged,2*numInputs+5:2*numInputs+4) = tmp(:,2)';
end
end % end per-pattern processing of results on if
end % end for loop to attempt to train a pattern
end % end for loop on bias sign
results = results(1:numConverged,:);
results = sortrows(results,2); % sort on number of steps to converge
rate = numConverged/sampleSize;
cs = num2str(rate*100);
message=strcat('Convergence rate ',cs);
message=strcat(message,'%');
disp(message);
mu=mean(results(:,2));
cs=int2str(mu);
message=strcat('Mean convergence ',cs);
disp(message);
disp(' ');

% for each pattern, display 1) average number of steps to converge,
% 2) average of volume estimates of attractors, 3) average Hamming
% distance between inputs of same class, and (optionally) 4) list of
% all input vectors, separated into the two classes, ordered by
% decreasing response
%
for i = 1 : numConverged % display the statistics
    disp(' ');
    cs = num2str(results(i,2));
    output = strcat('Average Number of Steps (weight changes) to converge= ',cs);
    disp(output);
end
disp(output);
cs = num2str(results(i,3));
output = strcat('Average Attractor Volume: ',cs);
disp(output);
cs = num2str(results(i,4));
output = strcat(...
   'Average Hamming Distance between equavalued patterns= ',cs);
disp(output);

250 if ((nargin == 3)&&(showPatternClasses==true))
   
   % display the classes of patterns
   
   disp(' %%%%%%%%%%%%%%%%% Individual input/raw response pairs %%%%%%%%%%%%%%%%%');
   disp('...
   %%%%%%%%%%%%%%%%% Negative raw responses correspond to output 0 %%%%%%%%%%%%%%%%%');
   for j = 1 : 2^numInputs
      cs = dec2bin(results(i,j+4),numInputs);
      output1 = strcat(' For input pattern= ',cs);
      cs = num2str(results(i,j+4+2^numInputs));
      output2 = strcat(' response was= ',cs);
      output = strcat(output1,output2);
      disp(output);
   end
   end
end

% plot graphical displays of results %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

270 h=figure(1);
   hist(convergenceResults(1:numConverged*numTrials),100);
   titleStr = strcat('Perceptron Weight Convergence Rate Study, inputs=');
   titleStr = strcat(titleStr,int2str(numInputs));
   titleStr = strcat(titleStr,', trials=');
   titleStr = strcat(titleStr,int2str(numTrials));
   title(titleStr);
   xlabel('Number of Weight Changes Required');
   ylabel('Number of Patterns');
   saveas(h,'Figure1.eps');
280 hold on
poisson = zeros ( ceil(max(results(:,2))), 2);
for i = 1 : ceil(max(results(:,2)))
    poisson(i,1) = i;
    poisson(i,2) = numInputs * rate * sampleSize * mu^i * exp(-mu)/factorial(i);
end

plot(poisson(:,1), poisson(:,2), 'LineWidth',4, 'LineStyle','--', ....
     'Color','k', 'Marker','*');

hold off

% figure 2 shows trajectory of weight vectors (for n=2),
% or "cloud of attractors" (for n=3)
if (numInputs == 2 | numInputs == 3)
    h = figure(2);
    if (numInputs == 2)
        limit = ceil(max(abs(min(min(min(history)))))),....
            max(max(max(history))));
        for i = 1 : numConverged
            showIndividualTrials = false;
            if (showIndividualTrials)
                paneNum = 1;
                for j = 1 : numTrials
                    subplot(3,3,paneNum);
                    tmp1 = history(i,j,1:attractors(i,j,numInputs + 2),1);
                    tmp2 = history(i,j,1:attractors(i,j,numInputs + 2),2);
                    plot (tmp1(1,:),tmp2(1,:));  % show the path of convergence
                    hold on;
                    plot(history(i,j,1,1),history(i,j,1,2), 'Marker', 'X', ....
                         'MarkerSize',10, 'Color','k');
                end
                hold on;
                grid on;
                axis([-limit limit -limit limit]);
                plot(attractors(i,j,1),attractors(i,j,2), 'Marker', 's', ....
                     'MarkerSize',10, 'Color','r');
                title(dec2bin(attractors(i,1,numInputs+1),2^numInputs));
                fName = strcat( 'f', int2str(i), 't', int2str(j), '.ps' );
                if (paneNum == 9 | i == numConverged)
                    fName = strcat( 'f', int2str(i-8), '-', int2str(i), '.ps' );
                end
                saveas(h,fName);
            end
        end
    end
end
if (paneNum == 9 & i ~= numConverged & numTrials ~= 9)
    paneNum = 0;
    close(h);
    h = figure(2);
end
end
paneNum = paneNum + 1;
hold off;
pause;
end  % loop on numTrials
close(h)
h = figure(2);
end  % end if showIndividualTrials
for j = 1 : numTrials;
    tmp1 = history(i,j,1:attractors(i,j,numInputs + 2),1);
    tmp2 = history(i,j,1:attractors(i,j,numInputs + 2),2);
    plot (tmp1(1,:),tmp2(1,:));  % show the path of convergence
    hold on;
    plot(history(i,j,1,1),history(i,j,1,2),'Marker','.','MarkerSize',10,'Color','k');
    hold on;
    plot(attractors(i,j,1),attractors(i,j,2),'Marker','x','MarkerSize',10,'Color','r');
    hold on;
end  % end loop on numTrials, all in one plot
grid on;
axis([-limit limit -limit limit]);
ml = strcat('...');
Outputs = ',dec2bin(attractors(i,numTrials+1,1),2*numInputs));
m2 = strcat('...Ave. Steps to Conv. = '...);
num2str(attractors(i,numTrials+1,2)));
title(strcat(ml,m2));
% plot the weight vector line attractors
if (i <= 7)
    w1=-limit:1/10:limit;
    w2=1;
    plot(w1,w2,'-k');
    w2=1-w1;
    plot(w1,w2,'-k');
end
w1=1;
w2 = limit : 1 / 10 : limit;
plot ( w1 , w2 , ' - - k ' ) ;

else
    w1 = limit : 1 / 10 : limit;
    w2 = - 1 ;
    plot ( w1 , w2 , ' - - k ' ) ;
    w2 = - 1 - w1 ;
    plot ( w1 , w2 , ' - - k ' ) ;
    w1 = - 1 ;
    w2 = limit : 1 / 10 : limit;
    plot ( w1 , w2 , ' - - k ' ) ;
end
% save figures
if ( i == 1 )
    saveas ( h , ' Figure 2 . eps ' ) ;
elseif ( i == 2 )
    saveas ( h , ' Figure 3 . eps ' ) ;
    pause
elseif ( i == 3 )
    saveas ( h , ' Figure 4 . eps ' ) ;
elseif ( i == 4 )
    saveas ( h , ' Figure 5 . eps ' ) ;
elseif ( i == 5 )
    saveas ( h , ' Figure 6 . eps ' ) ;
elseif ( i == 6 )
    saveas ( h , ' Figure 7 . eps ' ) ;
elseif ( i == 7 )
    saveas ( h , ' Figure 8 . eps ' ) ;
elseif ( i == 8 )
    saveas ( h , ' Figure 9 . eps ' ) ;
elseif ( i == 9 )
    saveas ( h , ' Figure 10 . eps ' ) ;
elseif ( i == 10 )
    saveas ( h , ' Figure 11 . eps ' ) ;
elseif ( i == 11 )
    saveas ( h , ' Figure 12 . eps ' ) ;
elseif ( i == 12 )
    saveas ( h , ' Figure 13 . eps ' ) ;
elseif ( i == 13 )
    saveas ( h , ' Figure 14 . eps ' ) ;
else
    saveas(h,'Figure15.eps');
end
    hold off;
end

% loop on numConverged for numInputs = 2
% plot the attractor cones from inequalities
h = figure(3);
x = [−limit 0 1 1];y = [1 1 0 −limit]; fill(x,y,[.9 .9 .9]);
hold on;
410 axis([−limit limit −limit limit]);
grid on;
saveas(h,'Figure16.eps');
hold off;

h = figure(3);
x = [1 1 1];y = [1 1 0]; fill(x,y,[.9 .9 .9]);
hold on;
axis([−limit limit −limit limit]);
grid on;
420 saveas(h,'Figure17.eps');
hold off;

h = figure(3);
x = [1 1 limit];y = [−limit 0 −limit+1]; fill(x,y,[.9 .9 .9]);
hold on;
axis([−limit limit −limit limit]);
grid on;
saveas(h,'Figure18.eps');
hold off;
430

h = figure(3);
x = [1 1 limit];y = [0 1 1 −limit+1]; fill(x,y,[.9 .9 .9]);
hold on;
axis([−limit limit −limit limit]);
grid on;
saveas(h,'Figure19.eps');
hold off;

h = figure(3);
440 x = [−limit+1 0 −limit+1];y = [1 1 limit]; fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,'Figure20.eps');
hold off;

h = figure(3);
x = [-limit+1 0 1 1];y = [limit 1 1 limit];fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,'Figure21.eps');
hold off;

h = figure(3);
x = [1 1 limit];y = [limit 1 1];fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,'Figure22.eps');
hold off;

h = figure(3);
x = [-limit -1 -1];y = [-1 -1 -limit];fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,'Figure23.eps');
hold off;

h = figure(3);
x = [-1 -1 0 limit];y = [-limit -1 -1 -limit];fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,'Figure24.eps');
hold off;

h = figure(3);
x = [limit -1 0 limit];y = [limit -1 -1];fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,’Figure25.eps’);
hold off;

h = figure(3);
x = [-limit -1 -1 -limit]; y = [-1 -1 0 limit -1]; fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,’Figure26.eps’);
hold off;

h = figure(3);
x = [-limit -1 -1]; y = [limit -1 0 limit]; fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,’Figure27.eps’);
hold off;

h = figure(3);
x = [-1 -1]; y = [0 -1 -1]; fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,’Figure28.eps’);
hold off;

h = figure(3);
x = [-1 -1 0 limit]; y = [limit 0 -1 -1]; fill(x,y,[.9 .9 .9]);
hold on;
axis([-limit limit -limit limit]);
grid on;
saveas(h,’Figure29.eps’);
hold off;

else  
  % show "attractor cloud", n=3
  limit = ceil(max(abs( min(min(min(attractors)))))},...
\[
\max(\max(\max(\text{attractors})))
\]

\text{paneNum} = 1;

\text{for } i = 1 : \text{numConverged}
\begin{align*}
\text{subplot}(3,3,\text{paneNum});
\text{scatter3(\text{attractors}(i,:),1,\text{attractors}(i,:),2,...
\text{attractors}(i,:),3,20,'.'});
\text{grid on;}
\text{axis([-2 2 -2 2 -2 2]);}
\text{title(strcat('O = ',dec2bin(\text{attractors}(i,\text{numTrials}+1,1),...\text{2*\text{numInputs}}),',S2C=',\text{num2str(\text{attractors}(i,\text{numTrials}+1,2)));}
\text{if (paneNum == 9 | i == \text{numConverged})}
\text{fName = strcat('f','int2str(i-8),'-','int2str(i),'.ps');}
\text{saveas(h,fName);}
\text{if (paneNum == 9 & i \neq \text{numConverged})}
\text{pause}
\text{paneNum = 0;}
\text{close(h);}
\text{h = figure(2);}
\text{end}
\end{align*}

\text{end}
\text{paneNum = paneNum + 1;}
\text{pause(1);}
\text{end}
\text{end} \% \text{if numInputs == 2}
\text{end} \% \text{if numInputs is 2 or 3}

\begin{align*}
\text{function } y = \text{binArray2Dec( binArray )} \\
y = 0; \\
\text{length} = \text{size(binArray,2);} \\
\text{for } i = 1 : \text{length} \\
\text{y = y + 2^((\text{length}-i)*\text{binArray}(i));} \\
\text{end}
\end{align*}

\begin{align*}
\text{function } y = \text{hammingDistance( binArray1, binArray2)} \\
y = 0; \\
\text{length} = \text{size(binArray1,2);}
\end{align*}
for i = 1 : length
    y = y + xor(binArray1(i), binArray2(i));
end

% this function returns an array of points randomly selected from arrayLen
% dimensional cube of side length 2*gridSize; uses normal distribution
function y = randnGrid (gridSize, arrayLen)
    y = randn(1, arrayLen);
    for i = 1 : arrayLen
        tmp = randn(1);
        if (tmp < 0)
            y(i) = y(i) + ceil(2*gridSize*tmp);
        else
            y(i) = y(i) + floor(2*gridSize*tmp);
        end
    end

end

% this function returns an array of points randomly selected from arrayLen
% dimensional cube of side length 2*gridSize; uses uniform distribution
function y = randGrid (gridSize, arrayLen)
    y = 2*gridSize*(rand(1, arrayLen) -.5);
References


