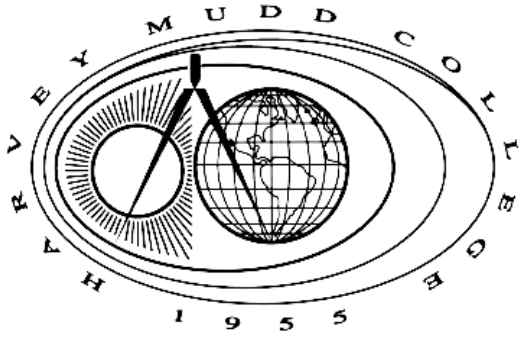
2000

Brownian Motion and Planar Regions: Constructing Boundaries from h-Functions

Otto Cortez
Harvey Mudd College



Brownian Motion and Planar Regions: Constructing Boundaries from h-Functions

by

Otto Cortez, '00

Lesley Ward, Advisor

Advisor: _____

Committee Member: _____

May 2000

Department of Mathematics

HARVEY MUDD COLLEGE

Abstract

Brownian Motion and Planar Regions: Constructing Boundaries from h-Functions

by Otto Cortez, '00

May 2000

In this thesis, we study the relationship between the geometric shape of a region in the plane, and certain probabilistic information about the behavior of Brownian particles inside the region. The probabilistic information is contained in a function $h(r)$, called the harmonic measure distribution function.

Consider a domain Ω in the plane, and fix a basepoint z_0 anywhere inside this domain Ω. Imagine lining the boundary of this domain with fly paper and releasing a million fireflies at the basepoint z_0. The fireflies wonder around inside this domain randomly until they hit a wall and get stuck in the fly paper. What fraction of these fireflies are stuck within a distance r of their starting point z_0? The answer is given by evaluating our h-function at this distance; that is, it is given by $h(r)$.

In more technical terms, the h-function gives the probability of a Brownian first particle hitting the boundary of the domain Ω within a radius r of the basepoint z_0. This function is dependent on the shape of the domain Ω, the location of the basepoint z_0, and the radius r.

The big question to consider is: How much information does the h-function contain about the shape of the domain's the boundary? It is known that an h-function cannot uniquely determine a domain, but is it possible to *construct* a domain that generates a given h-function? This is the question we try to answer.

We begin by giving some examples of domains with their h-functions, and then some examples of sequences of converging domains whose corresponding h-functions also converge to the correct h-function. In a specific case, we prove that artichoke domains converge to the wedge domain, and their h-functions also converge. Using another class of approximating domains, circle domains, we outline a method for constructing bounded domains from possible h-functions $f(r)$. We prove some results about these domains, and we finish with a possible approach for a proof of the convergence of the sequence of domains constructed.

TABLE OF CONTENTS

List of Figures **iii**

Chapter 1: Introduction **1**

1.1 Definition 1

1.2 Examples Using Discs 2

1.3 Previous Results and Useful Definitions and Theorems 5

1.4 Main Problem 8

1.5 Outline of Thesis 8

Chapter 2: A Model Example: Wedges and Artichokes **9**

Chapter 3: Approximations Using Circle Domains **14**

3.1 Scheme For Constructing Bounded Domains 16

3.2 New Labeling of Arcs 17

3.3 The Length of the Arc of Smallest Radius Goes to Zero 18

3.4 The Harmonic Measure of Each Arc Goes to Zero 21

3.5 Comments About Possible Domains 22

3.6 Other Ways to Choose Step Heights 23

3.7 Behavior of Circle Domains 24

3.8 Computing h-Functions of Circle Domains 25

Chapter 4: Numerical Results **26**

4.1 Decreasing Arc Lengths 26

4.2 Mappings 30

Chapter 5: Non-Uniqueness of Bounded Domains **33**

5.1 Scheme for Bounded Non-Unique Domains 33

Chapter 6: Strategy for Showing Convergence of Domains **35**

Chapter 7: Further Work **37**

Bibliography **38**

Appendix A: integrat.cc **40**

Appendix B: 6circle2.cc **43**

Appendix C: 3arcs_4.cc **50**

LIST OF FIGURES

1.1 The unit circle and its h-function. 3

1.2 The disc with an off-center basepoint z_0. 4

1.3 The h-function of the disc with an off-center basepoint. 5

2.1 The artichoke domain Ω_n. 10

2.2 The wedge domain Ω. 11

2.3 The intermediate domain G_n. 12

2.4 The domains R_1 and R_2. 13

3.1 An example of a circle domain. 15

3.2 Example of the new arc labeling. 17

3.3 Some important domains to consider. 20

3.4 The domain Ω_n''. 21

3.5 These are some impossible domains for continuous functions $f(r)$. . . 22

3.6 The h-function of domain Ω_2'. 24

3.7 The change in the h-function of domain Ω_2'. 25

4.1 The symmetry of step heights taken by averaging. 27

4.2 The domain Ω_0. 28

4.3 The domain Ω_1. 29

4.4 The domain Ω_2. 30

4.5 The space R for a two-arc circle domain. 31

4.6 The space R for a three-arc circle domain. 32

5.1 The domains Γ_n. 34

6.1 Possible bounding domains for the domain we are looking for. 35

6.2 Possible h-functions for the domains we are interested in. 36

ACKNOWLEDGMENTS

I would like to thank Professors Jorge Aarao and Byron Walden for their help in discussing the problem, and Marie Snipes for the work she did the year before. A very special thanks to Professor Ward for all the help she gave, the hours of discussion, and all the support she has given.

Chapter 1

Introduction

In this thesis we consider the relationship between the two-dimensional information of the shape of a domain's boundary and the one-dimensional information given by the h-function of this domain. The main problem considered here is that of constructing a domain given a possible h-function. We begin with some definitions and examples.

1.1 Definition

There are two equivalent definitions of the h-function we wish to consider. The first deals with Brownian particles and lends the problem some physical intuition. The second uses a more traditional definition of harmonic measure, the solution of a Dirichlet problem.

Definition 1.1.1. Consider a domain Ω in the extended complex plane whose boundary consists of a finite number of disjoint Jordan curves and/or arcs. Inside this domain, there is a basepoint z_0. Let $r > 0$ and define $E_r = \overline{B(z_0, r)} \cap \partial\Omega$. The *harmonic measure distribution function*, or h-function, at a radius r is the probability that a Brownian particle starting at the basepoint z_0 inside the domain Ω first hits the boundary of its domain within a radius r of z_0, that is, the probability that the particle hits E_r before it hits any other part of the boundary. It is a function of the basepoint z_0, the domain Ω, and the radius r.

Definition 1.1.2. Let Ω be a domain in the extended complex plane whose boundary consists of a finite number of disjoint Jordan curves. Let $r > 0$, and define $E_r = \overline{B(z_0, r)} \cap \partial\Omega$. The *harmonic measure* of E_r is given by evaluating at z_0 the solution of the following Dirichlet problem:

$$\begin{aligned} \Delta u &= 0 \quad \text{in } \Omega; \\ u &= \begin{cases} 1 & \text{on } E_r, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

The h-function is the collection of solutions to this problem for fixed z_0 and for all r.

The traditional notation for the harmonic measure distribution function for a domain Ω with basepoint z_0 at a radius r is $\omega(z_0, E_r, \Omega)$, where E_r is defined as above. We use this notation, and we also use the shorter form $h_\Omega(r)$.

1.2 Examples Using Discs

Perhaps the simplest domain to consider is the unit disc with the basepoint at the center. The h-function is simple. For radii less than 1, $h(r) = 0$ because no boundary has been included. For radii greater than or equal to 1, $h(r) = 1$ because all of the boundary is included, see Figure 1.1. As an aside, the harmonic measure of any segment of the boundary is simply the arc length of this segment over 2π (the fraction of the total arc length of the unit circle) since Brownian particles have an equal probability of hitting any part of the unit circle. For disjoint segments, one would sum the arc lengths and put them over 2π.

A more complicated domain is the disc with an off-center basepoint shown below. Here, d is the distance from the basepoint to the nearest point on the boundary and D is the furthest distance from the basepoint to a point on the boundary. Throughout this thesis, d and D keep these definitions. For $r > d$ but near d, one would expect the h-function to increase rapidly since Brownian particles are much more likely to

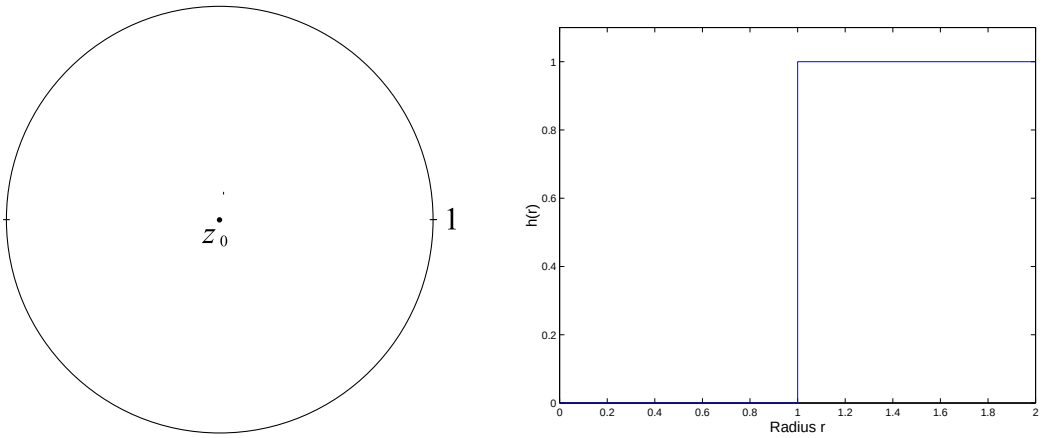


Figure 1.1: The unit circle and its h-function.

hit parts of boundaries nearest to where they are released. As r increases to D, the h-function should increase more slowly.

For a simple domain like this, one can calculate the h-function explicitly. Harmonic measure is a conformal invariant. To calculate the h-function of this domain we map it to a domain in which we know the h-function, such as the upper half plane. The region E_r is the region $\overline{B(z_0, r)} \cap \partial\Omega$, the intersection of the disk of radius r centered at z_0 and the boundary $\partial\Omega$ of the domain Ω. This is the region we are interested in. The function $h(r)$ is equal to the probability of a Brownian particle hitting this region E_r before it hits any other part of $\partial\Omega$. To calculate the h-function we map this domain to the upper half plane, keeping track of the endpoints of E_r.

For this calculation, we can assume the center of the circle is at the origin, its radius is 1, and z_0 lies on the horizontal axis. We can use the conformal map

$$\Phi(z) = i\frac{(1+z_0)(z-1)}{(z_0-1)(1+z)} \tag{1.1}$$

to map the off-center basepoint domain to the upper half plane. Using the law of cosines, we can see the endpoints of E_r are at

$$z_* = \exp\left[\pm i \cos^{-1}\left(\frac{1+z_0^2-r^2}{2z_0}\right)\right]. \tag{1.2}$$

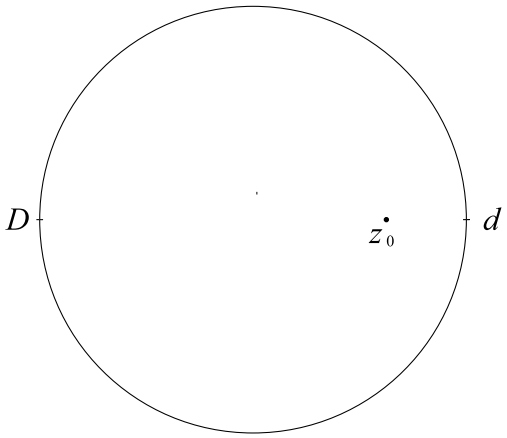


Figure 1.2: The disc with an off-center basepoint z_0.

A short calculation shows that these points map to

$$\Phi(z_*) = \mp \frac{1+z_0}{z_0-1} \tan\left(\frac{1}{2}\cos^{-1}\left(\frac{1+z_0^2-r^2}{2z_0}\right)\right), \tag{1.3}$$

and the basepoint z_0 maps to $\Phi(z_0) = i$. Now, the harmonic measure in the upper half plane is given by $1/\pi$ times the angle of sight between the basepoint, in this case i, and the endpoints of the segment on the boundary we are interested in, in this case $\Phi(z_*)$. (This we know by mapping the unit circle domain conformally to the upper half plane, and calculating the upper half plane's h-function.) Thus, we have a formula for the h-function of the off-center basepoint disc domain, and after some calculation we obtain

$$h(r) = \frac{2}{\pi}\arctan\left[\frac{D}{d}\sqrt{\frac{r^2-d^2}{D^2-r^2}}\right]. \tag{1.4}$$

The graph of this function is shown in Figure 1.3. For this example, $d = 20$ and $D = 80$. As we expected, the h-function increases rapidly for values of r slightly greater than d and less rapidly for middle values of r. As r gets close to D, it increases rapidly again. One would suspect this is because the rate at which boundary is being added as r increases also increases here. The h-function is always zero for values

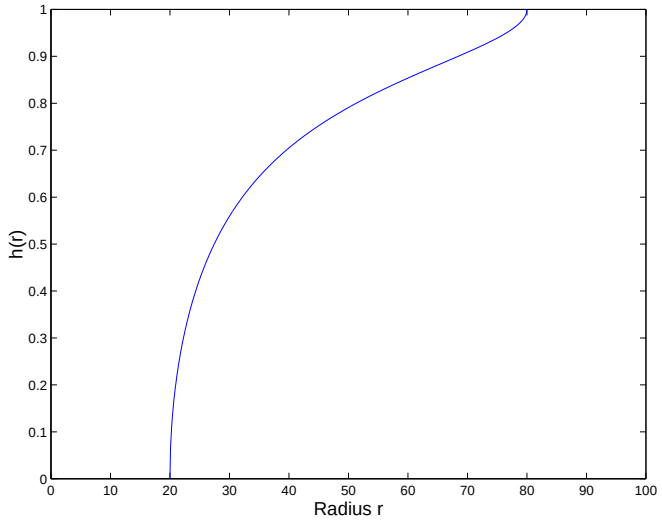


Figure 1.3: The h-function of the disc with an off-center basepoint.

of $r \leq d$ since no boundary is inside the disc of radius r. For bounded domains, the h-function is identically one for $r \geq D$ since all the boundary is included in the circle of radius r centered at the basepoint z_0. For non-bounded domains the h-function approaches one asymptotically. For $d \leq r \leq D$ the function is monotonically increasing.

Calculating the h-function for domains, even simple domains, is an excellent exercise in geometry.

1.3 Previous Results and Useful Definitions and Theorems

This thesis is a continuation of the work done by Marie Snipes, Professor Ward's previous research student. She obtained some significant results. The first of these which deals with circle domains, are defined in Chapter 3.

Theorem 1.3.1 (Snipes). *Let $f(r)$ be an arbitrary increasing step function with n steps, $f(d) = 0$ and $f(D) = 1$. Then there exists a circle domain Ω whose h-function is equal to $f(r)$.*

Furthermore, she shows that once the number of arcs is chosen and the positions of their midpoints are fixed there is continuous a one-to-one and onto mapping from the set of arc lengths in circle domains to the set of harmonic measures of these arcs. Thus, a step function can uniquely determine a circle domain. Snipes also showed that, given a bounded domain satisfying some additional conditions, one can approximate the domain with a sequence of converging circle domains, and their h-functions also converge to the proper h-function.

Other work has also been done by Professors Lesley Ward and Byron Walden. Their work includes some results on the non-uniqueness of domains, and on bounds on h-functions for $r \approx d$.

The rest of this section is a list of useful definitions and theorems taken from texts by Garnett and Marshall, and Pommerenke.

Theorem 1.3.2 (Garnett and Marshall, Ex. 21, Appendix B). *Suppose Ω is a finitely connected Jordan domain and suppose that $E \subset \{z \in \overline{\Omega} : |z| \geq R\}$ is a finite union of arcs on $\partial\Omega$. Let $z_0 \in \Omega$ with $|z_0| \leq r_0 < R$. Suppose that $\{z : |z| = r_0\}$ and $\{z : |z| = R$ are in the same component of the complement of $\Omega \cap \{z : r_0 < |z| < R\}$. Suppose further that $J_r \subset \{z \in \Omega : |z| = r\}$ separates z_0 from E, $r_0 < r < R$, and let $r\Theta(r)$ be the length of J_r. Then*

$$\omega(z_0, E, \Omega) \leq \frac{8}{\pi} \exp\left(-\pi \int_{r_0}^{R} \frac{dr}{r\Theta(r)}\right), \tag{1.5}$$

if $\Theta(r)$ is measurable.

Definition 1.3.1 (Carathéodory kernel convergence). Let $w_0 \in \mathbb{C}$ be given, and let G_n be domains with $w_0 \in G_n \subset \mathbb{C}$. We say that $G_n \to G$ as $n \to \infty$ with respect to w_0 in the sense of *kernel convergence* if

1. either $G = \{w_0\}$, or G is a domain $\neq \mathbb{C}$ with $w_0 \in G$ such that some neighborhood of every $w \in G$ lies in G_n for large n, and

2. for $w \in \partial G$ there exist $w_n \in \partial G_n$ such that $w_n \to w$ as $n \to \infty$.

Theorem 1.3.3 (Carathéodory kernel theorem). *Let f_n map $\mathbb{D}$ conformally onto G_n with $f_n(0) = w_0$ and $f_n'(0) > 0$. If $G = \{w_0\}$ let $f(z) \equiv 0$, otherwise let f map $\mathbb{D}$ conformally onto G with $f(0) = w_0$ and $f'(0) = 0$. Then, as $n \to \infty$, $f_n \to f$ locally uniformly in $\mathbb{D} \Leftrightarrow G_n \to G$ with respect to w_0.*

Definition 1.3.2 (Local connectivity). The closed set $A \subset \mathbb{C}$ is *locally connected* if for every $\epsilon > 0$ there exists a $\delta > 0$ such that for any two points a, b in A with $|a - b| < \delta$, we can find a path in A connecting a and b whose diameter is less than ϵ.

Definition 1.3.3 (Uniform local connectivity). A sequence of sets $\{A_n\}$ is *uniformly locally connected* if for every $\epsilon > 0$ there exists $\delta > 0$ independent of n such that for any two points a_n, b_n in A_n with $|a_n - b_n| < \delta$, we can find a path in A_n connecting a_n and b_n whose diameter is less than ϵ.

Theorem 1.3.4. *Let f map $\mathbb{D}$ conformally onto the bounded domain G. Then the following four conditions are equivalent:*

1. *f has a continuous extension to $\overline{\mathbb{D}}$;*

2. *∂G is a curve, that is $\partial G = \{\varphi(\zeta) : \zeta \in \mathbb{T}\}$ with continuous φ;*

3. *∂G is locally connected;*

4. *$\mathbb{C} \setminus G$ is locally connected.*

Theorem 1.3.5. *Let f_n map $\mathbb{D}$ conformally onto G_n with $f_n(0) = 0$. Suppose that there exist R_0 and R such that $B(0, R_0) \subset G_n \subset B(0, R)$, and that $\mathbb{C} \setminus G_n$ is uniformly locally connected. If $f_n(z) \to f(z)$ as $n \to \infty$ for each $z \in \mathbb{D}$, then the convergence is uniform in $\overline{\mathbb{D}}$.*

1.4 Main Problem

In a collection of research problems in complex analysis [3], K. Stephenson posed some questions about domains and their h-functions. All the questions dealt with the bigger question of how much information about the shape of a domain's boundary is encoded in its h-function. The question we consider here is:

> Given a possible h-function, can one construct a domain which generates this h-function?

We have made some progress in answering this question.

1.5 Outline of Thesis

We begin with an example of converging domains. In Chapter 2 we show that artichoke domains converge to the wedge domain, in the sense of Carathéodory kernel convergence, and that the h-functions of these domains also converge to the h-function of the wedge domain. In Chapter 3, we outline a scheme for constructing bounded domains from possible h-functions. we show that for functions that are continuous and have $f(d) = 0$, the arc at radius d must shrink to a point. We also outline some of the behavior of circle domains. There may be a way to calculate the h-functions of circle domains. It would seem that as the approximations to the function $f(r)$ get better, the arcs of circle domains decrease to the proper lengths. Chapter 4 shows some numerical examples of this. In this chapter, we also plot the map from the space of arc lengths to the space of harmonic measures. All known examples of non-uniqueness of domains deal with unbounded or multiply connected domains. In Chapter 5 we outline a method for constructing examples of bounded, non-unique, simply connected domains with the same h-function. In Chapter 6 we give a possible approach to a proof of the convergence of domains generated by the scheme in Chapter 3. Work remains to be done on this proof.

Chapter 2

A Model Example: Wedges and Artichokes

In this chapter we give an example of a sequence of converging domains, $\Omega_n \to \Omega$, generating a sequence of h-functions which converge to the correct h-function, $h_n(r) \to h(r)$. In general, if one knows the domain Ω satisfying certain properties, one can construct an approximating sequence of domains which converges to Ω and whose h-functions also converge. I show this for the specific case of the wedge domain and artichoke domains.

Let Ω_n be a sequence of domains with a fixed outer circle of radius D centered at the basepoint $z_0 = 0$, and $2^n + 1$ line segments beginning at radius d and stretching radially outward to the outer circle. Here $0 < d < D$. The boundary of the domain Ω_0 consists of the outer circle and two radial line segments. The first is located on the positive real axis. The second is located at an angle ϕ from the positive real axis and begins at the same distance d from z_0. For $n \geq 1$, the boundary consists of these first two radial line segments and $2^n - 1$ other line segments which divide the angle ϕ into 2^n equal sections; see Figure 2.1. We call these Ω_n *artichoke domains* since the disc with inward projections resembles the base of an artichoke.

Let Ω be the domain with a boundary consisting of the circle of radius D centered at the basepoint z_0, which has a wedge cut out of angle ϕ and inner radius of d; see Figure 2.2. As $n \to \infty$, it should be clear that the domains Ω_n converge to the domain Ω, at least in the rough sense that the domains Ω_n begin to look more and more like the domain Ω. We want to show that the h-functions for these domains also converge pointwise, that is, $h_{\Omega_n}(r) \to h_\Omega(r)$ for each $r > 0$.

Theorem 2.0.1 (Artichoke Convergence Theorem). *The sequence of artichoke*

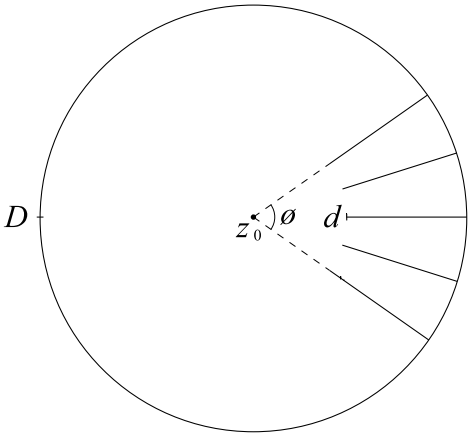


Figure 2.1: The artichoke domain Ω_n.

domains $\{\Omega_n\}$ *converges to the wedge domain* Ω *in the sense of Carathéodory kernel convergence, and the sequence of h-functions* $\{h_{\Omega_n}(r)\}$ *converges to* $h_\Omega(r)$ *pointwise.*

Proof: To verify that the sequence of artichoke domains $\{\Omega_n\}$ converges to the wedge domain Ω in the sense of Carathéodory convergence (Definition 1.3.1) we may use w_0=0. The first condition holds automatically since the Ω_n's are nested and decreasing. The second conditions holds automatically also for boundary points w in the outer circle or the radial segments in the boundary of Ω, since these points also lie in all of $\partial\Omega_n$. It also holds for all w in the inner circular arc in the boundary of Ω, since as n increases, any point on this arc can be approximated arbitrarily closely by the tips of radial line segments in $\partial\Omega_n$.

To show that $h_n(r) \to h(r)$ pointwise, we will use the intermediate domains G_n. Define the domain G_n to be the intersection of Ω_n with the wedge domain of angle ϕ, outer radius D, and inner radius $r_n' = d + \delta_n$; see Figure 2.3. As n increases, then δ_n decreases by a factor of two for each n. First we must show that for each $r > 0$, $|h_{\Omega_n}(r) - h_{G_n}(r)| \to 0$ as $n \to \infty$.

Let us define the two domains R_1 and R_2. The boundary of domain R_1 consists of two radial lines, one starting at d and extending to D on the real axis, and the

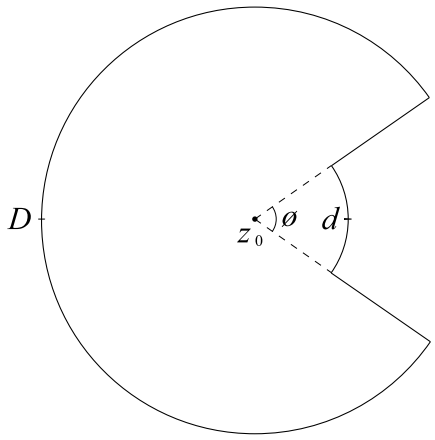


Figure 2.2: The wedge domain Ω.

other at an angle α. The basepoint is $z_0 = 0$. The second domain R_2 has the same boundary except that there is an arc joining the two radial lines at a radius of $d + \delta$. Let E in R_1 be the section of boundary within a radius $d + \delta$ of z_0, on the inside of the channel. Let E in R_2 be the inside of the channel in R_2 including the arc joining the two radial lines. See Figure 2.4.

Lemma 2.0.1. *The difference between the probability of hitting E in R_1 and the probability of hitting E in R_2, from the basepoint z_0, approaches zero exponentially as the channel width α decreases. Specifically:*

$$|P_{R_1}(hit\ E) - P_{R_2}(hit\ E)| \leq \frac{8}{\pi}\left(\frac{d+\delta-x_0}{d-x_0}\right)^{-\frac{\pi}{2}\cot\left(\frac{\alpha}{2}\right)}. \tag{2.1}$$

Proof: The proof of this is very similar to the proof shown in [9] and uses an estimate for harmonic measure from [5] shown in Section 1.3. This estimate says that

$$\omega(z_0, F, \Omega) \leq \frac{8}{\pi}\exp\left[-\pi\int_d^{d+\delta_n}\frac{dx}{2(x-x_0)\tan\frac{\alpha}{2}}\right], \tag{2.2}$$

where $E = \{z \in \partial\Omega : |z| > d + \delta\}$. Also, this $\omega(z_0, E, \Omega)$ is equivalent to the probability of hitting E^c from the arc a distance $d + \delta$ from z_0. The right hand side of (2.2) evaluates to the right hand side of (2.1). □

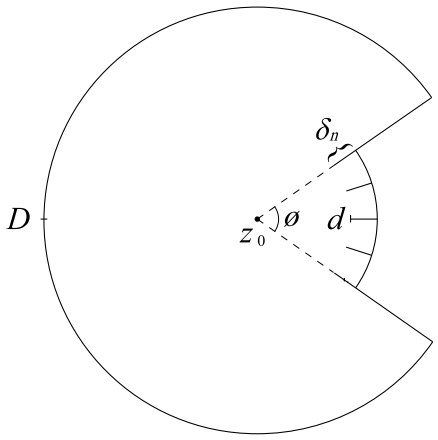


Figure 2.3: The intermediate domain G_n.

Now we wish to show that this result remains true for a collection of channels such as in our domains G_n. Let b_i be the arc of radius $d+\delta$ between the i^{th} and $(i+1)^{\text{th}}$ spike.

$$\begin{aligned} P(\text{cross } \cup b_i, \text{hit } E^c) &= \sum_{i=0}^{2^n-1} P(\text{cross } b_i, \text{hit } E^c) \\ &= \sum_{i=0}^{2^n-1} P(\text{cross } b_i) P(\text{hit } E^c \text{ from } b_i) \\ &\leq \sum_{i=0}^{2^n-1} P(\text{cross } b_i) \frac{8}{\pi} \left(\frac{d+\delta-x_0}{d-x_0} \right)^{-\frac{\pi}{2}\cot\left(\frac{\alpha}{2}\right)} \\ &\leq \frac{8}{\pi} \left(\frac{d+\delta-x_0}{d-x_0} \right)^{-\frac{\pi}{2}\cot\left(\frac{\alpha}{2}\right)}. \end{aligned} \tag{2.3}$$

This implies that $|h_{\Omega_n}(r) - h_{G_n}(r)| \to 0$ as $n \to \infty$.

Finally, we want to show that $h_{G_n}(r) \to h_{\Omega}(r)$ as $n \to \infty$. Notice that the Riemann map f_n which takes G_n to the unit disk is not one-to-one, since G_n is not a Jordan domain. Instead, f_n is two-to-one on parts of the boundary, specifically on the spikes. It is however possible to form a well-defined, continuous inverse f_n^{-1} in a manner very similar to that described in [9]. Since f^{-1} and f_n^{-1} are well defined, then

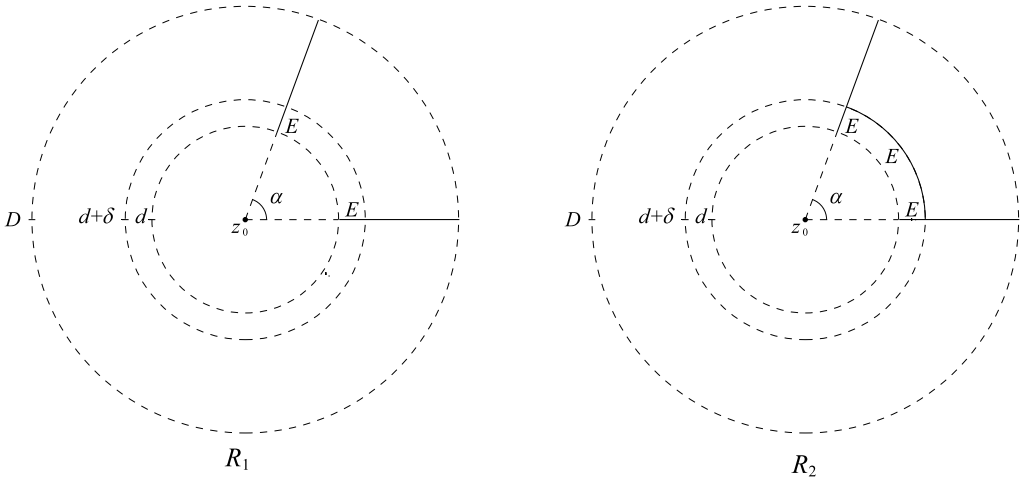


Figure 2.4: The domains R_1 and R_2.

the boundaries ∂G_n and $\partial\Omega$ are locally connected. The rest of the proof is as in [9] except for showing that ∂G_n is uniformly locally connected. For this we let $\epsilon, \delta > 0$. Let ϕ_n be the angle between adjacent radial lines. Let d_n be the length of each spike. See Figure 2.3. Let a_n and b_n be two points in ∂G_n such that $|a_n - b_n| < \delta$. Choose N large enough so that for $n > N$, $d_n < \epsilon/3$ and $\phi_n(r + d_n) < \epsilon/3$. Now a possible path along the boundary ∂G_n from point a_n to b_n would be down one spike, across the arc, and up the other spike, and this path has a diameter less than ϵ. Thus, ∂G_n is uniformly locally connected. □

Chapter 3

Approximations Using Circle Domains

The previous chapter gave an example of a sequence of converging domains, $\Omega_n \to \Omega$, whose corresponding sequence of h-functions, $\{h_n(r)\}$, also converges. Moreover, this sequence converges to the right function, that is $h_n(r) \to h(r)$, where $h(r)$ is the h-function of the domain Ω. In the example, the wedge domain was approximated using an artichoke domain; see Figures 2.2 and 2.1. A result similar to this can be achieved using circle domains [9].

Definition 3.0.1. A *circle domain* is a domain that is bounded by a circle and contains concentric arcs within this circle. It is bounded by a circle of radius D. The center of this circle is the basepoint z_0. We often take z_0 to be 0. Inside this circle are concentric boundary arcs centered at the horizontal which goes through the center of the circle, the basepoint z_0. The midpoint of the nearest arc is placed a distance d from z_0 along the horizontal. The other arcs are centered at dyadic points in the interval (d, D).

A circle domain is shown in Figure 3.1. From this figure, one can easily imagine that a sequence of circle domains will converge to the wedge domain, at least in the sense that they begin to look the same. Because circle domains are not simply connected, it is necessary to construct intermediate domains in a similar way as we did for the artichoke domains in Chapter 2. These intermediate domains converge to the wedge domain in the sense of Carathéodory kernel convergence.

As with many domains, there are no analytic formulas for the h-functions of artichoke domains or circle domains. The major difficulty with artichoke domains

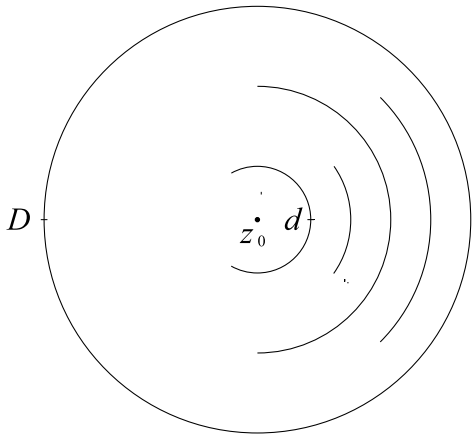


Figure 3.1: An example of a circle domain.

is that they have boundary spikes. There is a greater difficulty with circle domains because these are not simply connected. We do not have conformal mappings of these domains to domains in which we know the h-function. This problem may be overcome with artichoke domains since they are at least simply connected, so they can be conformally mapped to the disc or upper half plane. However, one would still need an explicit Riemann mapping in order to calculate the h-function.

There are two properties which make circle domains much more attractive than artichoke domains in trying to construct boundaries from h-functions. The first of these is that the h-functions of circle domains are step functions. Any circle of radius r about z_0 either includes a boundary arc or it does not meet that arc. As r increases from zero, at first no boundary is included, so $h(r) = 0$ for $r \leq d$. When $r = d$ a lot of boundary is included all of a sudden. So $h(r)$ jumps to a height h_d. As r continues to increase toward the radius of the next arc, r^*, the h-function stays constant because no new boundary is being included. When $r = r^*$, the h-function jumps up again because a lot of new boundary is being added instantaneously. So the h-functions of circle domains are step functions. The shape of the h-functions of artichoke domains are unknown. We could try numerical techniques to approximate

these h-functions and get some intuition about their shape, but this is time consuming and computationally intensive.

The second property is that there is a continuous, one-to-one correspondence between the arc lengths of arcs in a circle domain and the heights of the steps in the h-function. That is to say, if we let D be the space of circle domains with n arcs at specified radii $r_1, \ldots, r_n$, with arc midpoints on the positive real axis, and let R be the space of all possible h-functions of the circle domains D, then there exists a mapping $F : D \to R$ that is continuous, one-to-one and onto. The space D consists of all n-tuples $(x_1, \ldots, x_n)$ such that $x_i \in [0, 2\pi]$ for $i = 1, \ldots, n$. The space R is the space of all n-tuples $(y_1, \ldots, y_n)$ where $0 \leq y_1 \leq \cdots \leq y_n \leq 1$ [9, Theorem 2.1]. This is a strong result for circle domains, and we do not have an analogous, continuous bijection for artichoke domains.

In creating a scheme for constructing bounded domains, it would seem that circle domains would be more convenient. The following is a simple scheme along these lines.

3.1 Scheme For Constructing Bounded Domains

Let $f(r)$ be a continuous, monotonically increasing function on the interval $I = [d, D]$. Also, let $f(r) = 0$ for $r \leq d$ and $f(r) = 1$ for $r \geq D$. We can divide the interval $[d, D]$ into 2^n dyadic intervals as follows:

$$I_j = \left[d + j\frac{D-d}{2^n}, d + (j+1)\frac{D-d}{2^n} \right), \ 0 \leq j < 2^n. \tag{3.1}$$

We can approximate the function $f(r)$ with a step function $h_n(r)$. We choose $h_n(r)$ so it has jumps at dyadic points $d + j(D-d)/2^n$ in the interval I. We let $h_n(r)$ equal the average of $f(r)$ over the interval I_j, for $r \in I_j$. As $n \to \infty$, $h_n(r) \to f(r)$ pointwise. The question that remains is whether or not the domains which generate the $h_n(r)$'s as h-functions also converge?

To prove that these domains do converge, one would have to show that the length of each arc $A^n_{j,m}$ that appears in a domain Ω_m stabilizes as more and more arcs appear on either side of it (see Section 3.2 for an explanation of notation). We prove some results along these lines.

3.2 New Labeling of Arcs

Now we introduce a new labeling for the arcs, $A^n_{j,m}$, where Ω_n is the circle domain with 2^n arcs which generates the step function $h_n(r)$, j is the arc number in the circle domain Ω_m, where j increases with the distance from z_0, and Ω_m is the domain in which the arc at radius $r_{j,m} = d+(j-1)\frac{D-d}{2^m}$ first appears; see Figure 3.2. The domain Ω_0 would only contain one arc, $A^0_{1,0}$, at the same radius d as the arc labeled $A^2_{1,0}$ in Ω_2. The lengths of $A^0_{1,0}$ and $A^2_{1,0}$ might be different. The domain Ω_1 would contain the previous arc and the one labeled $A^2_{2,1}$. In the domain Ω_2 the arcs $A^2_{2,2}$ and $A^2_{4,2}$ would be added.

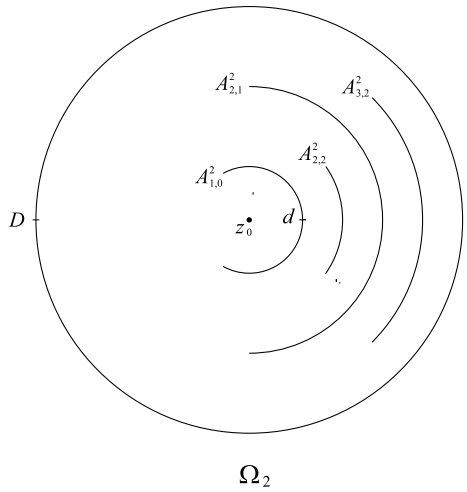


Figure 3.2: Example of the new arc labeling.

This labeling allows us to relate an arc to a specific step in the function $h_n(r)$,

the step beginning at $r_{j,m}$. An interesting note about these steps is that for $n > m$, $h_n(r_{j,m})$ is strictly decreasing as n increases if the heights of the steps of $h_n(r)$ are given by the averages of the function $f(r)$ over the intervals I_j. This is because $f(r)$ is increasing, and each $h_n(r)$ is right-continuous.

3.3 The Length of the Arc of Smallest Radius Goes to Zero

Theorem 3.3.1 (First Arc). *Let $f(r)$ be a monotonically increasing continuous function such that $f(d) = 0$ and $f(D) = 1$, and let $h_n(r)$ be a step function approximation of $f(r)$ as described below. Then the magnitude of the first arc approaches zero, $|A^n_{1,0}| \to 0$, as $n \to \infty$.*

Proof: We can approximate the function $f(r)$ using a series of step functions we call $h_n(r)$. Each of the step functions $h_n(r)$ has 2^n steps. The j^{th} step in each of these has a height equal to the average value of $f(r)$ over the interval

$$I = \left[d + j\frac{D-d}{2^n}, d + (j+1)\frac{D-d}{2^n} \right) \tag{3.2}$$

covered by this step, where $0 \leq j < 2^n$. That is

$$h_n\left(d + j\frac{D-d}{2^n} \right) = \frac{2^n}{D-d} \int_{I_j} f(r)\, dr. \tag{3.3}$$

By [9, Thm. 2.1] we know that each of these step functions is generated by a circle domain Ω_n of the following form. The boundary of this domain consists of concentric arcs centered at a basepoint $z_0 = 0$ and enclosed by a circle of radius D, also centered at z_0. In Ω_n, we can choose all boundary arcs to have midpoints with the same argument. For simplicity, we require these midpoints to lie on the positive real axis. With these constraints, the domain is unique. We know that each of the steps in $h_n(r)$ pertains to a boundary arc in this domain Ω_n. Specifically, the height of the jump in $h_n(r)$ at r is the harmonic measure of the arc at radius r seen from z_0. Further, the total height of $h_n(r)$ at r is the total harmonic measure of all the arcs at radii less than or equal to r, seen from z_0. Each Ω_n has 2^n arcs.

The first approximation of $f(r)$ is the step function $h_0(r)$. It has one step that covers the interval $I = [d, D)$ and has a height given by Equation (3.3); that is, its height is the average value of $f(r)$ on $[d, D)$. The domain Ω_0 is a domain containing one boundary arc at radius d inside the full circle of radius D. We continue to approximate the function by taking averages over successively smaller dyadic intervals inside this interval $[d, D)$. For example, the function $h_1(r)$ consists of two steps, the first over the interval $[d, d + \frac{D-d}{2})$ and the second over the interval $[d + \frac{D-d}{2}, D)$. The domain Ω_1 contains two boundary arcs, one at radius d and the other at radius $d + \frac{D-d}{2}$. Continuing in this way we see that as $n \to \infty$ the height of the first step in the function $h_n(r)$, that is the average of $f(r)$ over $[d, d + \frac{D-d}{2^n})$, approaches zero because $f(r)$ is continuous at d. This height is the harmonic measure, $\omega(z_0, A^n_{1,0}, \Omega_n)$, of the arc $A^n_{1,0}$ at radius d. This arc is the closest arc to our basepoint z_0. In general, the harmonic measure of the arc $A^n_{j,m}$, from z_0 in Ω_n, is given by

$$\omega(z_0, A^n_{j,m}, \Omega_n) = h_n(r_{j,m}) - h_n(r_{2^{n-m}(j-1),n}), \tag{3.4}$$

where $r_{j,m}$ and $r_{2^{n-m}(j-1),n}$ are the radii of the arcs $A^n_{j,m}$ and $A^n_{2^{n-m}(j-1),n}$ respectively (for $n > m$). For simplicity, we refer to the first arc, arc $A^n_{1,0}$, A_1. We want to show that $|A_1| \to 0$ as $n \to \infty$.

First we define the domains Ω_n, Γ_n and Ω'_n; see Figure 3.3. The domain Ω_n is the circle domain whose harmonic measure distribution function is given by the step function $h_n(r)$, the n^{th} approximation of $f(r)$. The domain Γ_n is identical to Ω_n except that the second arc, the arc $A^n_{2,n}$ (A_2 for short) at radius $r_{2,n} = d + \frac{D-d}{2^n}$, is longer. All other arcs being equal, there is now a higher probability of hitting arc A_2 in the domain Γ_n, and thus the harmonic measure of A_1 must decrease by the monotonicity of harmonic measure. In general, an increase in the length of an arc increases the harmonic measure of that arc while decreasing the harmonic measure of other parts of the boundary. We call this property the monotonicity of harmonic measure. So we have the inequality $\omega(z_0, A_1, \Gamma_n) < \omega(z_0, A_1, \Omega_n)$.

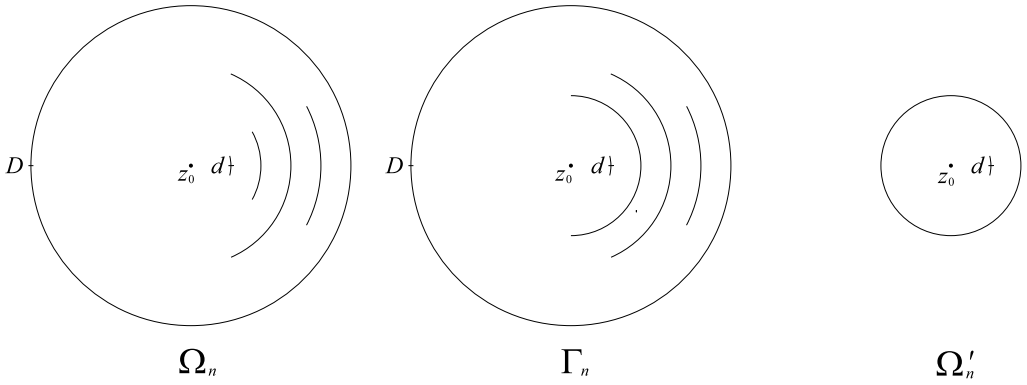


Figure 3.3: Some important domains to consider.

Let the domain Ω'_n be the limiting case of Γ_n where the arc at r_2 is a full circle. Then $\omega(z_0, A_1, \Omega'_n) < \omega(z_0, A_1, \Omega_n)$. We want to show the inequality

$$\frac{|A_1|}{2\pi} \leq \omega(z_0, A_1, \Omega'_n) < \omega(z_0, A_1, \Omega_n) \tag{3.5}$$

holds.

Define the domain Ω''_n to be a full disc of radius d, see Figure 3.4. For this it is convenient to think of the harmonic measure functions $\omega(z, A_1, \Omega'_n)$ and $\omega(z, A_1, \Omega''_n)$ as solutions to the Dirichlet problems for the domains Ω'_n and Ω''_n with boundary values of 1 on A_1 and 0 everywhere else. The arc A_1 has the same length in both domains. In Ω'' it forms part of the boundary circle. Recall that $\omega(z_0, A_1, \Omega''_n) = |A_1|/2\pi$. By [2], Corollary 4.7.6, we know that a solution to the problem exists in Ω'_n. By the maximum principle for harmonic functions we know that the maxima and minima of solutions to Dirichlet problems occur on the boundary. Thus, for all z in the domain Ω''_n, the strict inequality

$$0 < \omega(z, A_1, \Omega'_n) \tag{3.6}$$

holds.

Now consider the function $g(z) = \omega(z, A_1, \Omega'_n) - \omega(z, A_1, \Omega''_n)$ in the domain Ω''_n. This function $g(z)$ is harmonic in Ω'', equals zero only on the boundary arc A_1 and is

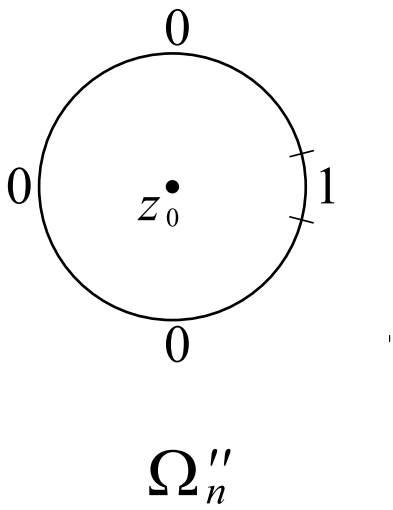


Figure 3.4: The domain Ω''_n.

positive for all other values on the boundary of Ω''_n. The maximum principle implies that $g(z) > 0$ everywhere inside Ω''_n. Therefore the inequality

$$\frac{|A_1|}{2\pi} = \omega(z_0, A_1, \Omega''_n) \leq \omega(z_0, A_1, \Omega'_n) \tag{3.7}$$

must hold. This gives us inequality (3.5).

Since $|A_1|/2\pi \leq \omega(z_0, A_1, \Omega_n) = h_n(d) \to 0$ as $n \to \infty$, it follows that $|A_1| \to 0$ as $n \to \infty$. □

3.4 The Harmonic Measure of Each Arc Goes to Zero

Lemma 3.4.1. *The harmonic measure of any individual arc approaches zero as* $n \to \infty$.

Proof: We know the harmonic measure of an arc $A^n_{j,m}$ is

$$\omega(z_0, A^n_{j,m}, \Omega_n) = h_n(r_{j,m}) - h_n(r_{j-1,n}) \tag{3.8}$$

where $r_{j,m}$ and $r_{j-1,m}$ are the radii of arcs $A^n_{j,m}$ and $A^n_{j-1,m}$ respectively. Notice that $h_n(r_{j,m})$ is decreasing as n increases for $n > m$, and has a greatest lower bound of $f(r_{j,m})$. Similarly, $h_n(r_{j-1,n})$ is increasing as n increases for $n > m$ and has a least upper bound of $f(r_{j,m})$. These two bounds are the same because f is continuous, and because of the definition of $h_n(r)$ in terms of averages of $f(r)$ over small intervals near $r_{j,m}$. Thus $\omega(z_0, A^n_{j,m}, \Omega_n) \to 0$ as $n \to \infty$. □

3.5 Comments About Possible Domains

The previous lemma implies that as n increases our circle domains are becoming smoother, that is, the length of the j^{th} arc is getting closer to the length of the $(j-1)^{\text{th}}$ arc and the $(j+1)^{\text{th}}$ arc. If the function $f(r)$ is continuous then the limiting domain should have no arc spiking out, for this would correspond to a discontinuity in the possible harmonic measure distribution function $f(r)$. Similarly, the limiting domain should not have parts of circles as boundaries since these would also correspond to discontinuities in the function $f(r)$. In both of these cases, this is because too much boundary is added at once. Thus the domains shown in Figure 3.5 are not possible for continuous functions $f(r)$.

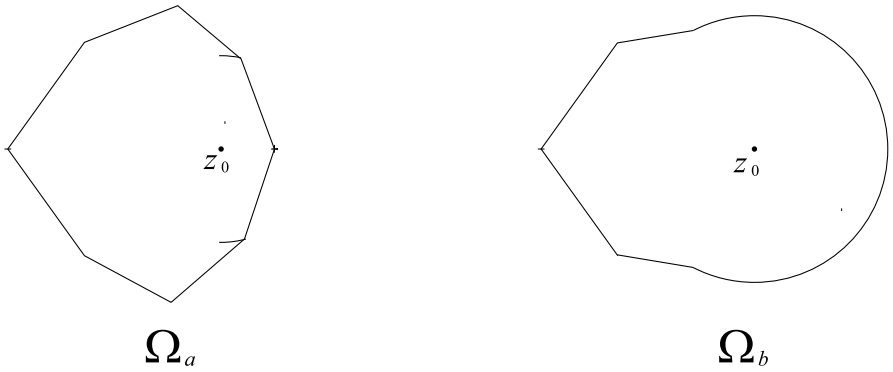


Figure 3.5: These are some impossible domains for continuous functions $f(r)$.

This method of constructing domains seems to imply that for any possible h-function $f(r)$ there is a bounded domain. Moreover, there is such a domain that is symmetric about the horizontal. This symmetry is a bit counterintuitive, and this property comes from an arbitrary decision in positioning the arcs inside our circle domains. The arcs do not have to be symmetric about the horizontal; their midpoints could be specified to lie at any given arguments. Choosing the horizontal was a decision made in the interest of simplicity.

3.6 Other Ways to Choose Step Heights

Choosing step heights for the function $h_n(r)$ which approximates a possible continuous h-function $f(r)$ to be the averages of this function over the intervals I_j (as in Equation (3.1)) was an arbitrary decision. Other possible ways to choose step heights would be to set them equal to the maximum, or minimum, of the function $f(r)$ over the intervals I_j. We would still have pointwise convergence, $h_n(r) \to f(r)$ for all $r > 0$.

The h-functions of these three methods arise from domains with slightly varying behavior. For example, if we use the minimum value of $f(r)$ over the interval I_j for the step heights of our functions $h_n(r)$, then we immediately see that the arc length of the first arc is zero. That is, the first arc is just a point. With respect to the first arc, taking the maximum value of $f(r)$ over the intervals I_j gives domains that behave much like those which generate step heights given by the average of $f(r)$ over the intervals I_j.

Taking the step heights to be averages leads to step heights which change at every iteration. That is, for a given radius $r_{j,m}$ the step height would decrease as n increases. For the other two methods, the height of the steps at these radii remains the same for $n > m$. This gives us a better grasp of what is going on with the domains because there is less movement in the step functions.

3.7 Behavior of Circle Domains

Consider a continuous, monotonic function $f(r)$ such that $f(d) = 0$ and $f(D) = 1$. Approximate this function with the step functions $h_n(r)$ whose step heights are the minimum values of $f(r)$ over the dyadic intervals I_j, for $j = 1, \ldots, 2^n$. Let $n = 2$; then the domain Ω_2 has four arcs. The first arc, $A^2_{1,0}$, has arc length zero, so it is a point. Assume for the moment that $|A^2_{2,1}| = 0$ also. Call this domain Ω'_2. Then the h-function of this domain appears as in Figure 3.6. Adding the arc $A^2_{2,1}$ would cause the h-function of the domain Ω'_2 to change as follows (see Figure 3.7).

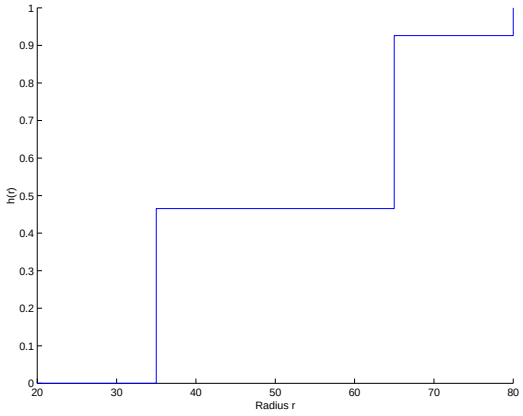


Figure 3.6: The h-function of domain Ω'_2.

The domain Ω'_2 would change to some other domain Ω^*_2 with $|A^2_{2,1}| > 0$. This must cause at least one of the arcs in domain Ω'_2 to decrease in arc length. This is because the height of $h_{\Omega^*_2}(r_{4,2})$ has not changed from $h_{\Omega'_2}(r_{4,2})$, while there has been new boundary added to arc $A^2_{2,1}$. Notice, the height of $h_{\Omega'_2}(r)$ at $r_{4,2}$ is the harmonic measure of all the arcs in domain Ω'_2, and it is too high. Thus, by the monotonicity of harmonic measure, at least one arc should decrease in length. The value of $h_2(r_{2,2})$ is now lower than it should be, so it would seem that $|A^2_{2,2}|$ needs to increase. By a similar argument, $|A^2_{4,2}|$ needs to decrease.

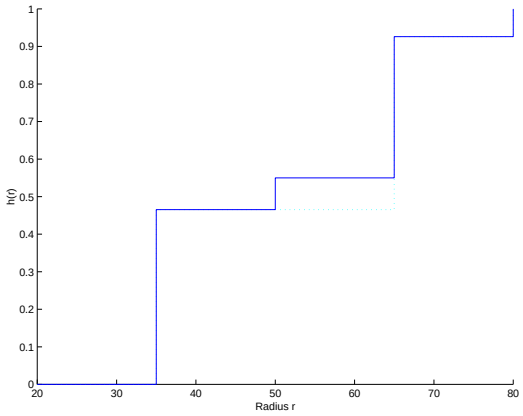


Figure 3.7: The change in the h-function of domain Ω_2'.

Similar arguments can be made for the cases where the step heights are taken to be the averages or maximum of $f(r)$ over the intervals I_j.

3.8 Computing h-Functions of Circle Domains

It may be possible to compute the h-functions of one-arc circle domains analytically. In an unpublished work by Byron Walden, he derives the h-function for two-slit domains. A slit domain is a domain in the extended complex plane with finite length slits of boundary along the real axis. It may be possible to find an explicit, conformal mapping from our one-arc circle domains to one of these domains, and thus have an analytic formula for the h-function of a circle domain. Computation of the modules of these domains would give some information on the feasibility of this approach, since the module is also a conformal invariant. See [1].

Chapter 4

Numerical Results

4.1 Decreasing Arc Lengths

Let $f(r)$ be a continuous, monotonic function such that $f(d) = 0$ and $f(D) = 1$, where $0 < d < D$. The functions $h_n(r)$ are step function approximations to $f(r)$. The height of each step in $h_n(r)$ comes from the average of the function $f(r)$ over some dyadic interval. Say the interval was $[a, b]$. Then in the next approximation this interval, would be broken up into two, $[a, a + (b - a)/2]$ and $[a + (b - a)/2, b]$. Moreover, each of the two step heights is the same distance from the step that came before it in the previous approximation since

$$\frac{1}{b-a}\int_a^b f(r)\,dr = \frac{1}{2}\left(\frac{1}{(b-a)/2}\int_a^{a+\frac{b-a}{2}} f(r)\,dr + \frac{1}{(b-a)/2}\int_{a+\frac{b-a}{2}}^b f(r)\,dr\right). \tag{4.1}$$

See Figure 4.1. That is, the average of f over $[a, b]$ is equal to the average of its averages on the two halves of $[a, b]$. In the example, the interval $[a, b] = [35, 65]$. We tried to use this fact to show that the magnitude of the arc nearest the base point z_0, $|A^n_{1,0}|$, is decreasing to zero instead of simply going to zero. (Notation is explained in Section 3.2.) Specifically, we were trying to use this to show that $|A^0_{1,0}| > |A^1_{1,0}|$. While we were not able to prove this rigorously, we have some numerical evidence to support this idea. It also seems that the other arcs are decreasing monotonically. The desired step heights were arrived at by integrating averages with `integrate.cc` (a C++ program which integrates averages of functions over specified intervals), and the magnitudes of the angles were estimated by using `6circle2.cc` (a C++ program started by Snipes which simulates Brownian particles in circle domains). The source

code for these is included in the appendices.

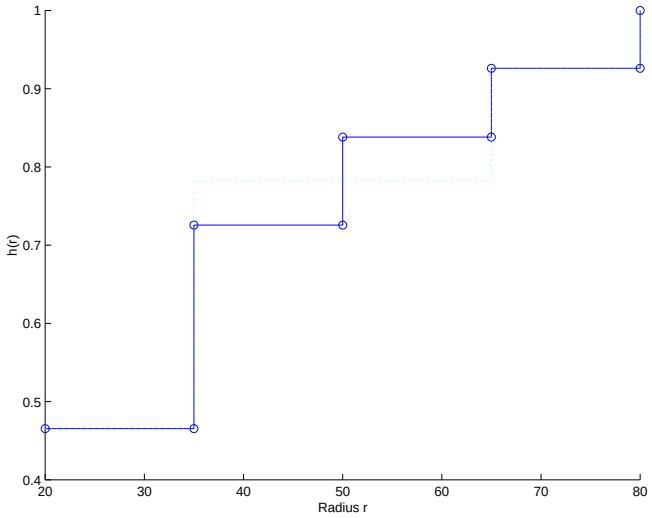


Figure 4.1: The symmetry of step heights taken by averaging.

The following tables and figures give numerical examples of approximating the h-function for the circle with the off-center base point domain shown in Figure 1.3. The h-function is given by Equation (1.4). The program `6circle2.cc` tests guesses for arc lengths needed to produce a given $h_n(r)$. In this way, we can estimate the lengths of the arcs in our circle domains. The circle domain used had an outer circle of radius $D = 80$ and the arc nearest the base point z_0 at the center of the circle domain is a distance $d = 20$. The first domain, Ω_0, has one arc a distance d away from z_0 with a magnitude of 0.42π. (Our convention is that this magnitude is half the angle subtended at z_0 by the arc.) The harmonic measure of this one arc is roughly the average of the function $f(r)$ over the interval $[d, D] = [20, 80]$. The domain is drawn in Figure 4.2. The dotted lines indicate the actual domain being approximated, that is, the domain with the off-center base point. The arcs extend past what is expected.

The domain Ω_1 has two arcs of boundary inside the circle of radius 80. These arcs are located at radii $r_{1,0} = d = 20$ and $r_{2,1} = 50$. The estimated magnitudes for

	$A^0_{1,0}$
Mag. (% of π)	0.42
$h_0(r_{1,0})$	0.7326
Actual Avg.	0.738899

Table 4.1: Numerical results from domain Ω_0.

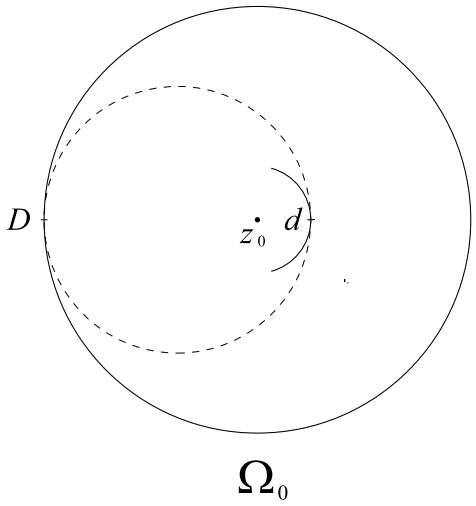


Figure 4.2: The domain Ω_0.

these arcs $A^1_{1,0}$ and $A^1_{2,1}$ are 0.315π and 0.72π respectively. Since there is some variance between the harmonic measure of these arcs and the actual harmonic measure desired, some more refinement can be made to these arc lengths. These arc lengths are quite close though, and as can be seen in Figure 4.3, the arcs are still too large.

The domain Ω_2 contains four boundary arcs inside the circle of radius 80. The two new arcs are located at radii $r_{2,2} = 35$ and $r_{4,2} = 65$. The arc lengths are $|A^2_{1,0}| = 0.245\pi$, $|A^2_{2,2}| = 0.505\pi$, $|A^2_{2,1}| = 0.651\pi$, and $|A^2_{3,2}| = 0.79\pi$. Again these are just estimates to the lengths of the arcs, as one can see by comparing the harmonic measures against the actual harmonic measures desired. Again, one can see in Figure

	$A^1_{1,0}$	$A^1_{2,1}$
Mag. (% of π)	0.315	0.72
$h_1(r_{i,j})$	0.5979	0.8984
Actual Avg.	0.595597	0.882201

Table 4.2: Numerical results from domain Ω_1.

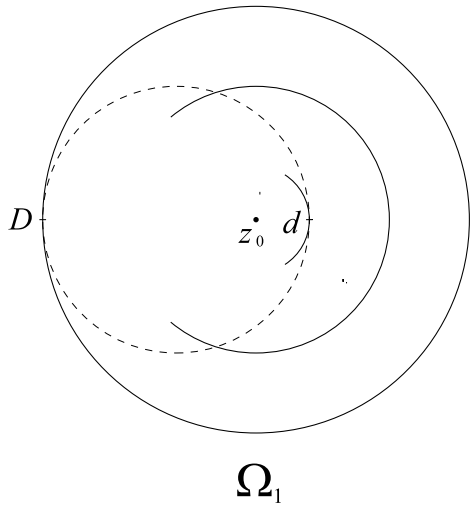


Figure 4.3: The domain Ω_1.

4.4, that these arc lengths are still too large.

It would seem the arcs are bounded below by the dotted circle in the figures above. Doing some calculations, we can calculate these bounds to be: $|A^n_{1,0}| \geq 0$, $|A^n_{2,2}| \geq 0.4429\pi$, $|A^n_{2,1}| \geq 0.5970\pi$, and $|A^n_{4,2}| \geq 0.7350\pi$. These numbers seem to support the idea that arc lengths should be decreasing, and perhaps monotonically decreasing.

	$A^2_{1,0}$	$A^2_{2,2}$	$A^2_{2,1}$	$A^2_{4,2}$
Mag. (% of π)	0.245	0.505	0.651	0.79
$h_2(r_{i,j})$	0.4656	0.7258	0.8423	0.9271
Actual Avg.	0.465423	0.725771	0.838257	0.926145

Table 4.3: Numerical results from domain Ω_2.

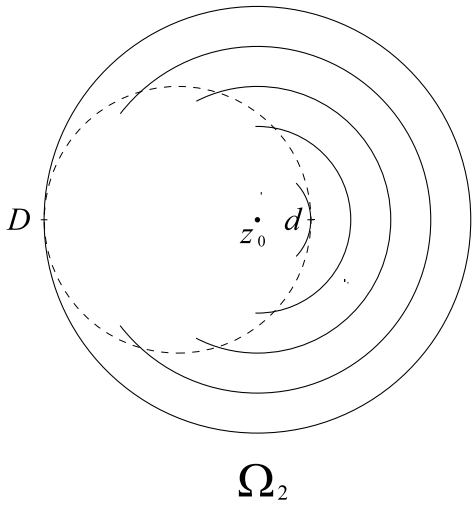


Figure 4.4: The domain Ω_2.

4.2 Mappings

The following figures show the mappings $D \to R$ discussed in Chapter 3. Only the R space is shown. Figure 4.5 shows the map for a two-arc circle domain with an outer circle of radius $D = 80$. The first arc is at radius $d = 20$ and the second arc is at radius 50. The points plotted represent the harmonic measures of the first (horizontal axis) and second (vertical axis) arcs, as the lengths of the arcs are increased from 0 to 2π in increments of $2\pi/10$. This map shows how strong the shadowing effect of the first arc is on the second arc. Figure 4.6 is the R space for the domain with a third

arc added at radius 35. The important detail here is to notice that the map in Figure 4.5 is a level set of this three dimensional map. Adding more arcs to circle domains only adds more dimensions to these maps. These maps are continuous. The real question is, does convergence in one infinite dimensional space imply convergence in another? (These figure were produced using the C++ program `3arcs_4.cc`, which is an edited version of a program written by Snipes to simulate Brownian motion inside circle domains.)

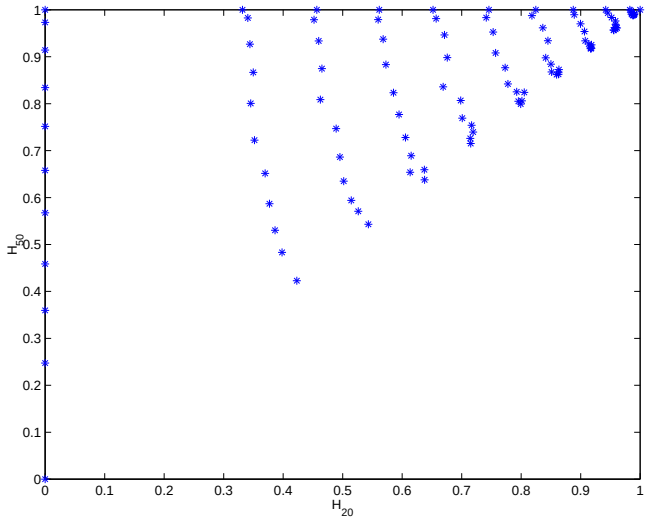


Figure 4.5: The space R for a two-arc circle domain.

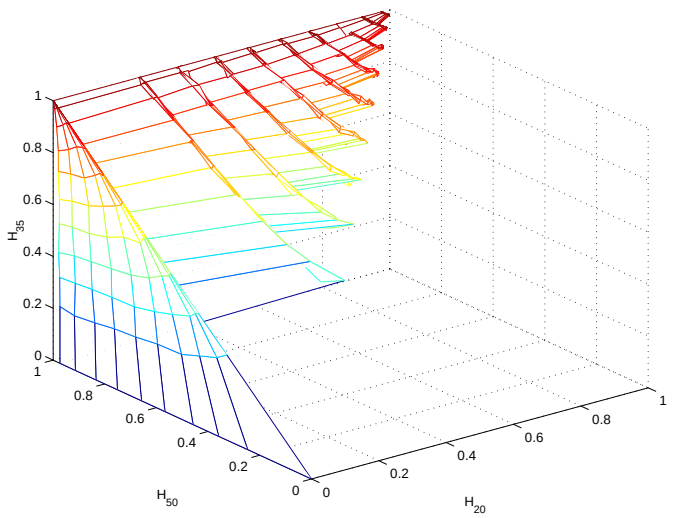


Figure 4.6: The space R for a three-arc circle domain.

Chapter 5

Non-Uniqueness of Bounded Domains

The method described in Section 3.1 would create bounded domains that are symmetric about a horizontal axis. With this method, it would seem that from any possible h-function $f(r)$ with $f(d) = 0$ and $f(D) = 1$ we could create a symmetric domain. Up to rotations, translations, and inversions, this is probably the only symmetric domain which would generate this function as its h-function. Examples of non-uniqueness of domains for a given h-function have been given in [10] and [9]. Examples in [10] have been of two unbounded domains with the same h-function or one unbounded domain with the same h-function as a bounded domain. The examples in [9] are of circle domains. While these are bounded, they are not simply connected.

In this chapter we try to find other bounded, simply connected domains with the same h-functions by giving a scheme for how we might find such domains.

5.1 Scheme for Bounded Non-Unique Domains

Consider a new type of circle domains Γ_n. These domains have an outer circle of radius D centered at the base point z_0, and also have concentric boundary arcs. The midpoints of these arcs are not along the horizontal going through the base point z_0 though. The midpoints are along a line beginning at a horizontal distance d from the base point z_0 and offset by an angle ϕ, see Figure 5.1. As with our previous circle domains, these also have a continuous, one-to-one and onto mapping from D to R [9, Theorem 2.1]. If we had a sequence of circle domains Ω_n which converged, and whose h-functions converged to the right function, then we would suspect we could

find a collection of Γ_n domains with the same h-functions, and that these would also converge. Once again, we are left with showing a sequence of converging h-functions is generated by a sequence of converging domains. And, at this stage, we do not have the estimates on harmonic measure necessary to complete this argument.

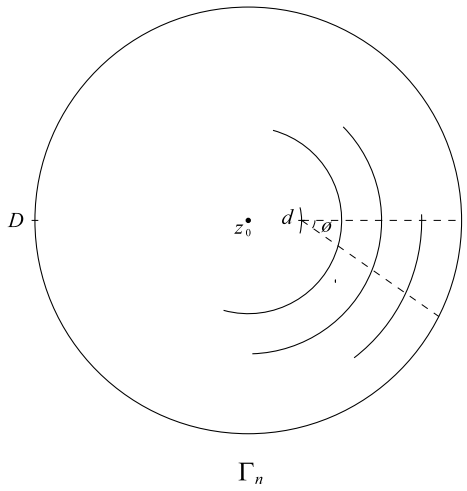


Figure 5.1: The domains Γ_n.

Chapter 6

Strategy for Showing Convergence of Domains

Suppose $f(r)$ is a continuous, monotonic function which is identically zero on the interval $(-\infty, d]$ and identically one on the interval $[D, \infty)$, where $0 < d < D$. We wish to construct a domain Ω whose h-function is $f(r)$. We begin by approximating the function $f(r)$. A close approximation of the function $f(r)$ is given by splitting the interval up into 2^n dyadic intervals. Define a step function, $h_n(r)$, which on each of the 2^n dyadic intervals is identically equal to the *minimum* value of $f(r)$ over that interval. We expect that the domain Ω_n which corresponds to the h-function $h_n(r)$ should be reasonably close to the domain we are looking for, for large n. Now, we can close off the tunnels in this domain by joining the endpoints of the arcs with straight line segments, as shown with domains Ω_n and Ω_n' below.

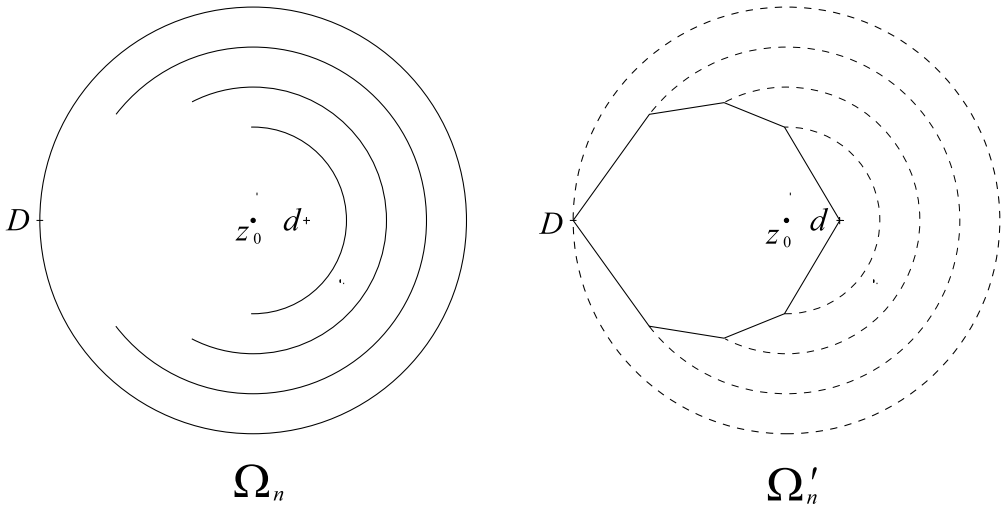


Figure 6.1: Possible bounding domains for the domain we are looking for.

If the function $f(r)$ is continuous over the entire interval $[d, D]$, then we wish to join the last arc, arc $A^n_{2^n,n}$ (see Section 3.2 for notation), to the circle of radius D at the point along the horizontal going through the basepoint z_0 at an angle of π from the center of the first arc, arc $A^n_{1,0}$. If $f(r)$ is not continuous at $r = D$, then a bound needs to be found for this last arc, and we could join this arc with the outer circle with radial line segments. We expect that $h_{\Omega_n}(r) \leq f(r) \leq h_{\Omega'_n}(r)$ for most r. This is because in order for a Brownian particle to hit one of the arcs it must first cross the lines between the two arcs, the solid lines in Figure 6.1. We expect the h-functions to look like those in the graphs in Figure 6.2. If this is true, we could now pin down the domain we are looking for by squeezing the function $f(r)$ between $h_{\Omega_n}(r)$ and $h_{\Omega'_n}(r)$. We expect that as n increases, the maximum distance between the functions $h_{\Omega_n}(r)$ and $h_{\Omega'_n}(r)$ goes to zero (i.e. $|h_{\Omega_n}(r) - h_{\Omega'_n}(r)| \to 0$). This is because, as n increases, the probability of getting very far into any tunnel falls off exponentially, as shown by the estimate of Garnett and Marshall [5]. Thus, the approximation to our limiting domain given by Ω'_n is not a bad one, but it is a bit too big. To remedy this the only solution seems to be that each of the arcs must decrease in arc length.

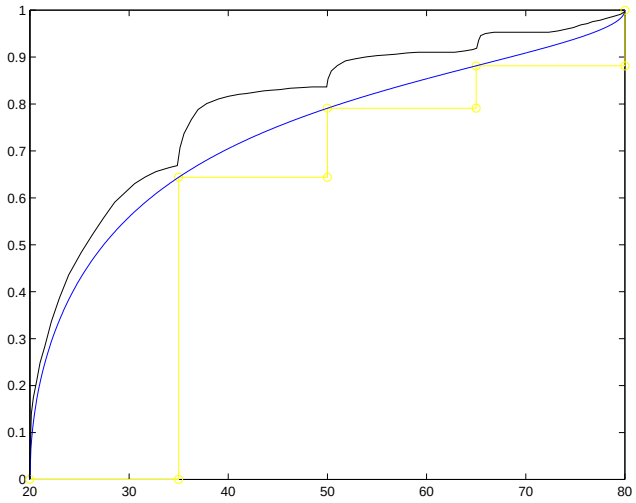


Figure 6.2: Possible h-functions for the domains we are interested in.

Chapter 7

Further Work

It seems plausible that the sequence of circle domains which would generate the sequence of functions $h_n(r)$, which in turn, approximate a possible h-function $f(r)$, would converge to a limiting domain with the proper h-function. Work still needs to be done in showing the domain discussed in Chapter 6 in fact do bound the function $f(r)$ from above. In particular, we would like to show that these domains do not generate h-functions with strange behavior like too many points of inflection. After this is shown, it should be possible to show the stability of arclengths for the arcs in our sequence of circle domains.

More work should also be done in the study of the maps discussed in Chapter 3 from the space of circle domains to the space of harmonic measures. It may also be possible to actually calculate the h-functions of some circle domains in the way discussed in Section 3.8.

BIBLIOGRAPHY

[1] Lars V. Ahlfors. *Conformal Invariants: Topics in Geometric Function Theory.* McGraw-Hill, 1973.

[2] Carlos Berenstein and Roger Gay. *Complex Variables: An Introduction.* Springer-Verlag, 1991.

[3] D.A. Brannan and W.K. Hayman. Research problems in complex analysis. *Bull. London Math. Soc. 21*, 1989.

[4] W. H. J. Fuchs. *Topics in the Theory of Functions of One Complex Variable.* Cornell University, 1967.

[5] Garnett and Marshall. *Harmonic Measure.* Chapter 3. In preparation.

[6] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, 1994.

[7] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 2nd edition, 1994.

[8] Christian Pommerenke. *Boundary Behavior of Conformal Maps.* Springer-Verlag, 1992.

[9] Marie Snipes. *Brownian Motion and the Shape of a Region's Boundary: Generating Step Functions from Circle Domains.* Harvey Mudd College, May 1999. Senior Thesis.

[10] Byron L. Walden and Lesley A. Ward. Distributions of harmonic measure for planar domains. *Proceedings of XVIth Rolf Nevanlinna Colloquium, Eds.: Laine/Martio*, pages 289–299, 1996.

Appendix A

INTEGRAT.CC

```
/*
 * Input:
 *    a, b: limits of integration.
 *    n: number of trapezoids.
 * Output:  Estimate of the integral from a to b of f(x)
 *    using the trapezoidal rule and n trapezoids, and then
 *    take average value over integral.
 *
 * Note:  f(x) is hardwired.
 */
#include<stdio.h>
#include<iostream.h>
#include<stdlib.h>
#include<math.h>


const double D = 80;

const double d = 20;

//const double M_PI = 3.141592653589793;
void Get_data(double &, double &, int &, int &);
double Trap(double, double ,int ,double);
double Average(double &, double &, int &, double &);
double simpleAverage(double, double, double);
void Output(double &, double &);

int main(int argc, char** argv) {
    double      a;           /* Left endpoint             */
    double      b;           /* Right endpoint            */
    double      b_also;
    double      a_also;
    int         n;           /* Number of trapezoids      */
    double      h;           /* Trapezoid base length     */
    double      integral;    /* Integral over my interval */
    double      average;
    int         parts;       /* Parts to break into */

    Get_data(a, b, n, parts);
    h = (b-a)/(((int)(n/parts))*parts);
    a_also = a;
```

```
    b_also = a + (b-a)/parts;
    average = 0;
    for (int i=0;i<parts;i++)
      {
integral = Trap(a_also, b_also, (int)(n/parts), h);
average =  simpleAverage(integral,a_also,b_also);
Output(a_also,average);
a_also = b_also;
b_also = b_also + (b-a)/parts;
      }
    cout << a_also << "\t" << 1 << "\n";
    return 0;
} /*  main  */


/*********************************************************************/
/* Function Get_data
 * Reads in the user input a, b, and n.
 * Input parameters:
 *     1.  int my_rank:  rank of current process.
 *     2.  int p:  number of processes.
 * Output parameters:
 *     1.  double* a_ptr:  pointer to left endpoint a.
 *     2.  double* b_ptr:  pointer to right endpoint b.
 *     3.  int* n_ptr:  pointer to number of trapezoids.
 * Algorithm:
 *     1.  Process 0 prompts user for input and
 *         reads in the values.
 *     2.  Process 0 sends input values to other
 *         processes.
 */
void Get_data(double &a_ptr, double &b_ptr, int &n_ptr, int &parts_ptr)
{
    cerr << "Enter a (left limit, double):";
    cin >> a_ptr;
    cerr << "Enter b (right limit, double):";
    cin >> b_ptr;
    //    cerr << "Enter n (partitions, int): ";
    //    cin >> n_ptr;
    n_ptr=1000000;
    cerr << "Enter number of parts (arcs, int):";
    cin >> parts_ptr;
    cerr << n_ptr << " partitions \n";
} /* Get_data */


/*********************************************************************/
double Trap(
          double  local_a   /* in */,
          double  local_b   /* in */,
```

```
          int    local_n   /* in */,
          double  h         /* in */)
{
    double integral;   /* Store result in integral  */
    double x;
    int i;

    double f(double x); /* function we're integrating */
    integral = (f(local_a) + f(local_b))/2.0;
    x = local_a;
    for (i = 1; i <= local_n-1; i++) {
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral*h;
    return integral;
} /*  Trap  */


/********************************************************************/
double f(double x)
{
  //return x*x;
      return ((2/M_PI)*atan((D/d)*sqrt((x*x-d*d)/(D*D-x*x))));
} /* f */

// Average uses Trap and divides by (b-a)
double Average(double &local_a, double &local_b,int &local_n,
               double  &h)
  {
    return (Trap(local_a,local_b,local_n,h)/(local_b - local_a));
  }

double simpleAverage(double sum, double x, double y)
  {
    return (sum/(y-x));
  }

void Output(double &integ, double &avg)
  {
    cout.precision(16);
    cout << integ << "\t" << avg << "\n";
  }
```

Appendix B

6circle2.cc

```
//Marie Snipes
//Otto Cortez (Summer 1999)
//Summer, 1998

/* This a program that starts a brownian particle at the origin of a
   circle domain and records when and where it hits the boundary of
   the domain.  We can specify the number of runs that the computer
   will execute.  In addition, this time, each arc keeps track of the
   number of times it has been hit, and the Display function prints
   this out at the end of the program.  This way, we do not have to
   fix the arc radii in order to count the hits.  */

#include<iostream.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

// create a class called Arc
class Arc
{
public:
  Arc(double, double);
  double CheckForBoundaryHit(double [2][2]);
  double Distance(void);
  double Angle(void);
  int numberOfBoundaryHits;
  double distance, angle;
};

//constructor for class Arc
Arc::Arc(double d, double a)
{
  distance = d;
  angle = a;
  numberOfBoundaryHits = 0;
}


//create a class called Node (this will be the linked list)

class Node
```

```
{
public:
  Node(double, double);
  Arc arc;
  Node *next; //next is a pointer to a Node.
};

//constructor for class Node
Node::Node(double distance, double angle) : arc(distance, angle)
{
  next = NULL;
}


const double stepsize = 1;
//const double M_PI = 3.141592653589793;
double PickTheta();
void Move(double [2][2], double);
double CheckWholeBoundary(Node *, double [2][2]);
double Arg(double, double);
int WithinBounds(double, double);
Node *Add_Arc(Node *start, double, double);
void Display_Arcs(Node *start);

int main( void )
{
  double theta;
  double coord[2][2] = {{0,0}, {0,0}};
  double arcAngle;  //note: arcangle varies between 0 and M_PI
  double arcRadius;
  double outerCircleRadius;
  int numberOfMoves;
  int numberOfArcs;
  srand(time(NULL));
  int numberOfRuns;
  cout.precision(16);
  //cerr << "What is the radius of the outer circle?";
  //cin >> outerCircleRadius;
  outerCircleRadius = 80;
  while (outerCircleRadius <= 0)
    {
      cout << "Invalid arc radius.  Please choose an outer circle radius > 0."
<< endl;
      cerr << "What is the radius of the outer circle?";
      cin >> outerCircleRadius;
    }

  Node *start = new Node(outerCircleRadius, M_PI);

  //cerr << "How many arcs are in the domain?";
```

```
  //cin >> numberOfArcs;
  numberOfArcs = 4;

  for (int j=0; j<numberOfArcs; j++) //create linked list of arcs
    {
      cerr << "What is the radius of arc " << j+1 << "?";
      cin >> arcRadius;
      while (arcRadius >= outerCircleRadius && arcRadius != 0 || arcRadius < 0)
{
  cout << "Invalid arc radius.  Please choose an arc radius < "
       << outerCircleRadius << " and > 0." << endl;
  cout << "What is the radius of arc " << j+1 << "?";
  cin >> arcRadius;
}

      //now we are sure that arcRadius < outerCircleRadius

      cerr << "What is its angle?";
      cin >> arcAngle;
      start = Add_Arc(start, arcRadius, M_PI*arcAngle);
    }

  //cerr << "How many runs?";
  //cin >> numberOfRuns;
  numberOfRuns = 10000;
  // start simulation
  for (int index = 0; index < numberOfRuns; index++)
    {
      double coord[2][2] = {{0,0}, {0,0}};
      for (numberOfMoves = 1; (numberOfMoves <= 10000000) &&
     (CheckWholeBoundary(start, coord) == 0); numberOfMoves++)
        {
  theta = PickTheta();
  Move(coord, theta);
        }
      if (index % 1000 == 0)
{
  cerr << "*";
}
    }
  cerr << "\n";
  Display_Arcs(start);
  return 0;
}

double PickTheta( void )
{
  double x, z;
  x = rand();
  z = 2*M_PI*(1 + x)/(RAND_MAX + 1);
```

```
  return z;
}

void Move(double coord[2][2], double theta)
{
  double dx, dy;
  dx = stepsize * cos(theta);
  dy = stepsize * sin(theta);
  //  cout << coord[0][0] << ", " << coord[0][1] << endl;
  coord[0][0] = coord[1][0];         // set "old x" to "new x"
  coord[0][1] = coord[1][1];         // set "old y" to "new y"
  coord[1][0] = coord[0][0] + dx;    // set "new x" to "old x + dx"
  coord[1][1] = coord[0][1] + dy;    // set "new y" to "old y + dy"
}

double CheckWholeBoundary(Node *start, double coord[2][2])
{
  Node *current;
  for(current = start; current != NULL; current = current->next)
    //(*current).next=current->next
    if(current->arc.CheckForBoundaryHit(coord) != 0)
      {
current->arc.numberOfBoundaryHits =
  current->arc.numberOfBoundaryHits + 1;
return current->arc.CheckForBoundaryHit(coord);
      }
  return 0;
}

double Arc::Distance()
{return distance;}

double Arc::Angle()
{return angle;}

double Arc::CheckForBoundaryHit(double coord[2][2])
{
  double oldDistanceFromOrigin, newDistanceFromOrigin, argold, argnew, m,
    temp1, temp2, ickyvar1, ickyvar2, x_a, x, y;
  int argoldInBounds, argnewInBounds;

  oldDistanceFromOrigin = sqrt((coord[0][0])*(coord[0][0]) +
       (coord[0][1])*(coord[0][1]));
  newDistanceFromOrigin = sqrt((coord[1][0])*(coord[1][0]) +
       (coord[1][1])*(coord[1][1]));

  if ((oldDistanceFromOrigin < distance && newDistanceFromOrigin < distance) ||
      (oldDistanceFromOrigin > distance && newDistanceFromOrigin > distance))
    return 0;
```

```
  /*if this is not true then we crossed the circle, in which case we
    need to see if where we crossed was actually within the angle of
    the arc.*/

  if(angle == M_PI)
    return distance;
  argold = Arg(coord[0][0], coord[0][1]);

  /*if we saved this from the last computation we might take less time
  to do the calculation, but we'd need more memory to store the value.
  Also applies to oldDistanceFromOrigin, etc*/

  argnew = Arg(coord[1][0], coord[1][1]);

  argoldInBounds = WithinBounds(angle, argold);
  argnewInBounds = WithinBounds(angle, argnew);

  if(newDistanceFromOrigin == distance)
    {
      if (argnewInBounds == 0)
return 0;
      else
return distance;
    }

  if ((oldDistanceFromOrigin < distance && newDistanceFromOrigin > distance) ||
      (oldDistanceFromOrigin > distance && newDistanceFromOrigin < distance))
    {

      if (argnewInBounds == argoldInBounds)
{
  if (argnewInBounds == 0)
    return 0;
  else
    return distance;
}
      else
{
  m = (coord[1][1] - coord[0][1])/(coord[1][0] - coord[0][0]);
  temp1 = m*coord[0][0] - coord[0][1];
  temp2 = 1 + m*m;

  ickyvar1 = m*temp1;
  ickyvar2 = sqrt(distance*distance*temp2 - temp1*temp1);

  x_a = (ickyvar1 + ickyvar2)/temp2;

  if( ((coord[0][0] < x_a) && (x_a < coord[1][0])) ||
      ((coord[1][0] < x_a) && (x_a < coord[0][0])) )
```

```
    x = x_a;

  /* In the case where x_1 = x_2, we crossed the arc
     vertically, in which case the args were equal, which we
     already took care of.*/

  else
    x = (ickyvar1 - ickyvar2)/temp2;

  y = sqrt(distance*distance - x*x);

  if (WithinBounds(angle, Arg(x, y)) == 0)
    return 0;
  else
    return distance;
}
    }
}

double Arg(double x, double y)  //Arg returns a value between 0 and Pi.
{
  double arg;
  arg = atan(y/x);
  if (x >= 0)
    {
      return arg;
    }
  if (x < 0)
    {
      if (y >= 0) return (M_PI + arg);
      else return(arg - M_PI);
    }
  return (-1*M_PI);
} // end of Arg

int WithinBounds(double arcangle, double argToCheck)
{

  if(fabs(arcangle) >= fabs(argToCheck))
    return 1;
  else
    return 0;
}

Node *Add_Arc(Node *start, double distance, double angle)
// returns a pointer to the start node while adding that arc to the
// list in order of how far it is from the origin.

{
  int count;
```

```
  Node *yptr, *previous, *current;
  yptr = new Node(distance, angle);

  //  Display_Arcs(start);

  for(count = 1, current = start, previous = NULL;
      (current != NULL) && (yptr->arc.Distance() > current->arc.Distance());
      count++, previous = current, current = current->next);

  if((*yptr).arc.Distance() == (*current).arc.Distance())
    {
      delete yptr;
      cout << "You tried to add an arc where you already had one." << endl;
      return start;
    }

  if (current == start) // add to front of an existing list
    {
      (*yptr).next = start;
      return yptr;
    }

  (*previous).next = yptr;  // add to the middle of an existing list
  (*yptr).next = current;

  return start;
}

void Display_Arcs(Node *start)
{
  int Sum=0;
  Node *current, *previous;
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      Sum=current->arc.numberOfBoundaryHits + Sum;
    }
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      cout << "distance: " << current->arc.Distance() << endl
 << "angle: " << current->arc.Angle() << endl
 << "number of boundary hits: " << current->arc.numberOfBoundaryHits
 << endl;
      cout << "Avg: " << ((current->arc.numberOfBoundaryHits) / Sum) << "\n\n";
    }
  cout << "\n\n Sum: " << Sum << "\n";

}
```

Appendix C

3arcs_4.cc

```
//Marie Snipes
//Otto Cortez (Spring 2000)

/* This program simulates Brownian motion inside circle domains with
    three arcs.  It adjusts the arc lengths to make the maps from the
    domain space to the harmonic measure space and outputs information
    in a form that can be used by matlab.*/


#include<iostream.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

// create a class called Arc
class Arc
{
public:
  Arc(double, double);
  double CheckForBoundaryHit(double [2][2]);
  double Distance(void);
  double Angle(void);
  int numberOfBoundaryHits;
  double average;
  void Reset(void);
  void editAngle(double);
  double distance, angle;
};

//constructor for class Arc
Arc::Arc(double d, double a)
{
  distance = d;
  angle = a;
  numberOfBoundaryHits = 0;
}


//create a class called Node (this will be the linked list)

class Node
```

```
{
public:
  Node(double, double);
  Arc arc;
  Node *next; //next is a pointer to a Node.
};

//constructor for class Node
Node::Node(double distance, double angle) : arc(distance, angle)
{
  next = NULL;
}


const double stepsize = 1;
const double D = 80;
const double d = 20;
const int MAX_ITER = 10;
//const double M_PI = 3.141592653589793;
double PickTheta();
void Move(double [2][2], double);
Node *GetInitialData(double &,int &,int &);
double CheckWholeBoundary(Node *, double [2][2]);
double Arg(double, double);
int WithinBounds(double, double);
Node *Add_Arc(Node *start, double, double);
void Simulate(Node *start,int &, int , int, int);
void Display_Arcs(Node *start, int , int, int);

int main( void )
{
  double outerCircleRadius;
  int numberOfArcs;
  int numberOfRuns;

  srand(time(NULL));
  cout.precision(16);
  Node *start = GetInitialData(outerCircleRadius,numberOfArcs,numberOfRuns);
  for (int iter1 = 0; iter1 <= MAX_ITER; iter1++)
    {
      for (int iter2 = 0; iter2 <= MAX_ITER; iter2++)
{
  for (int iter3 = 0; iter3 <= MAX_ITER; iter3++)
    {
      Simulate(start,numberOfRuns,iter1,iter2,iter3);
      Display_Arcs(start,iter1,iter2,iter3);
      cerr << iter1 << " " << iter2 << " " << iter3 << "\n";
    }
}
    }
```

```
  return 0;
} //end main



double PickTheta( void )
{
  double x, z;
  x = rand();
  z = 2*M_PI*(1 + x)/(RAND_MAX + 1);
  return z;
} // end of PickTheta()

void Move(double coord[2][2], double theta)
{
  double dx, dy;

  dx = stepsize * cos(theta);
  dy = stepsize * sin(theta);

  coord[0][0] = coord[1][0];         // "old x" = "new x"
  coord[0][1] = coord[1][1];         // "old y" = "new y"

  coord[1][0] = coord[0][0] + dx;    // "new x" = "old x + dx"
  coord[1][1] = coord[0][1] + dy;    // "new y" = "old y + dy"
} // end of Move()

// return 1 for hit, 0 for no hit.  Add 1 to number of hits to arc hit
double CheckWholeBoundary(Node *start, double coord[2][2])
{
  Node *current;
  for(current = start; current != NULL; current = current->next)
    //(*current).next=current->next)
    if(current->arc.CheckForBoundaryHit(coord) != 0)
      {
current->arc.numberOfBoundaryHits =
  current->arc.numberOfBoundaryHits + 1;
//return current->arc.CheckForBoundaryHit(coord);
return 1;
      }
  return 0;
} // end of CheckWholeBoundary()

double Arc::Distance()
{return distance;}

double Arc::Angle()
{return angle;}

double Arc::CheckForBoundaryHit(double coord[2][2])
```

```
{
  double oldDistanceFromOrigin, newDistanceFromOrigin, argold,
    argnew, m, temp1, temp2, ickyvar1, ickyvar2, x_a, x, y;
  int argoldInBounds, argnewInBounds;

  oldDistanceFromOrigin = sqrt((coord[0][0])*(coord[0][0])
       + (coord[0][1])*(coord[0][1]));
  newDistanceFromOrigin = sqrt((coord[1][0])*(coord[1][0])
       + (coord[1][1])*(coord[1][1]));

  if ((oldDistanceFromOrigin < distance && newDistanceFromOrigin < distance) ||
      (oldDistanceFromOrigin > distance && newDistanceFromOrigin > distance))
    return 0;


  //if this is not true then we crossed the circle, in which case we
  //need to see if where we crossed was actually within the angle of
  //the arc.

  if(angle == M_PI)
    {
      //cerr << "hit circle \n";
      return 1;
    }
  argold = Arg(coord[0][0], coord[0][1]);
  argnew = Arg(coord[1][0], coord[1][1]);

  argoldInBounds = WithinBounds(angle, argold);
  argnewInBounds = WithinBounds(angle, argnew);

  if(newDistanceFromOrigin == distance)
    {
      if (argnewInBounds == 0)
return 0;
      else
return 1;
    }

  if ((oldDistanceFromOrigin < distance && newDistanceFromOrigin > distance) ||
      (oldDistanceFromOrigin > distance && newDistanceFromOrigin < distance))
    {
      if (argnewInBounds == argoldInBounds)
{
  if (argnewInBounds == 0)
    return 0;
  else
    return 1;
}
      else
{
```

```
  m = (coord[1][1] - coord[0][1])/(coord[1][0] - coord[0][0]);
  temp1 = m*coord[0][0] - coord[0][1];
  temp2 = 1 + m*m;

  ickyvar1 = m*temp1;
  ickyvar2 = sqrt(distance*distance*temp2 - temp1*temp1);

  x_a = (ickyvar1 + ickyvar2)/temp2;

  if( ((coord[0][0] < x_a) && (x_a < coord[1][0])) ||
      ((coord[1][0] < x_a) && (x_a < coord[0][0])) )
    x = x_a;

  /* In the case where x_1 = x_2, we crossed the arc
     vertically, in which case the args were equal, which we
     already took care of.*/

  else
    x = (ickyvar1 - ickyvar2)/temp2;

  y = sqrt(distance*distance - x*x);

  if (WithinBounds(angle, Arg(x, y)) == 0)
    return 0;
  else
    return distance;
}
    }
  return 1;
} // end of Arc.CheckBoundaryHit()


double Arg(double x, double y)  //Arg returns a value between 0 and Pi.
{
  double arg;
  arg = atan(y/x);

  if (x >= 0)
    {
      return arg;
    }
  if (x < 0)
    {
      if (y >= 0) return (M_PI + arg);
      else return(arg - M_PI);
    }
  return (-1*M_PI);
} // end of Arg

// returns 1 if within arc, or 0 if outside arc
```

```
int WithinBounds(double arcangle, double argToCheck)
{

  if(fabs(arcangle) >= fabs(argToCheck))
    return 1;
  else
    return 0;
} // end of WithinBounds()

Node *Add_Arc(Node *start, double distance, double angle)
 // returns a pointer to the start node while adding that arc to the
 // list in order of how far it is from the origin.
{
  int count;
  Node *yptr, *previous, *current;
  yptr = new Node(distance, angle);

  //  Display_Arcs(start);

  for(count = 1, current = start, previous = NULL;
      (current != NULL) && (yptr->arc.Distance() > current->arc.Distance());
      count++, previous = current, current = current->next);

  if((*yptr).arc.Distance() == (*current).arc.Distance())
    {
      delete yptr;
      cout << "# You tried to add an arc where you already had one." << endl;
      return start;
    }

  if (current == start) // add to front of an existing list
    {
      (*yptr).next = start;
      return yptr;
    }

  (*previous).next = yptr;  // add to the middle of an existing list
  (*yptr).next = current;

  return start;
} // end of Node.AddArc()

void SetCoord(double startPt, double coord[2][2])
{
  coord[0][0] = 0;
  coord[0][1] = 0;
  coord[1][0] = startPt;
  coord[1][1] = 0;
} // end of SetCoord()
```

```
void Display_Arcs(Node *start, int iter, int iter2, int iter3)
{
  int Sum = 0;
  int SumArc20 = 0;
  int SumArc35 = 0;
  int SumArc50 = 0;
  Node *current, *previous;
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      if(current->arc.distance <= 20)
{
  SumArc20 = SumArc20 + current->arc.numberOfBoundaryHits;
}
      if(current->arc.distance <= 35)
{
  SumArc35 = SumArc35 + current->arc.numberOfBoundaryHits;
}
      if(current->arc.distance <= 50)
{
  SumArc50 = SumArc50 + current->arc.numberOfBoundaryHits;
}
      Sum=current->arc.numberOfBoundaryHits + Sum;
    }
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      if(current->arc.distance == 20)
{
  current->arc.average = (double)SumArc20 / (double)Sum;
}
      if(current->arc.distance == 35)
{
  current->arc.average = (double)SumArc35 / (double)Sum;
}
      if(current->arc.distance == 50)
{
  current->arc.average = (double)SumArc50 / (double)Sum;
}
    }
  if (iter == 0 && iter2 == 0 && iter3 == 0)
    {
      cout << "p = [";
      for (current = start, previous = NULL; current != NULL;
   previous = current, current = current->next)
{
  cout << current->arc.Distance() << ";\n";
}
      cout << "]; \n";
    }
```

```
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      if (current->arc.distance == 20)
{
  cout << "H1_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.average << ";\n";
  cout << "A1_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.angle << ";\n";
}
      else if (current->arc.distance == 50)
{
  cout << "H2_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.average << ";\n";
  cout << "A2_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.angle << ";\n";
}
      else if (current->arc.distance == 35)
{
  cout << "H3_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.average << ";\n";
  cout << "A3_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.angle << ";\n";
}
      else
{
  cout << "H4_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.average << ";\n";
  cout << "A4_" << iter << "_" << iter2 << "(" << (iter3+1) << ")="
       << current->arc.angle << ";\n"; }
    }
} //end Display_Arcs



Node *GetInitialData(double &radius,int &arcs,int &runs)
{
  double arcRadius;
  double arcAngle;  //note: arc angle varies between 0 and M_PI
  cerr << "What is the radius of the outer circle? \t";
  cin >> radius;
  cerr << radius << "\n";
  arcs = 3;
  cerr << "How many runs? \t";
  cin >>  runs;
  cerr << runs << "\n";
  while (radius <= 0)
    {
      cerr << "Invalid arc radius.  Please choose an outer"
           << "circle radius > 0.\n";
```

```
      cerr << "What is the radius of the outer circle?\t";
      cin >> radius;
    }
  Node *start = new Node(radius, M_PI);
  for (int j=0; j < arcs; j++) //create linked list of arcs
    {
      cerr << "What is the radius of arc " << j+1 << "?";
      cin >> arcRadius;
      while (arcRadius >= radius && arcRadius != 0 || arcRadius < 0)
{
  cerr << "Invalid arc radius.  Please choose an arc radius < "
       << radius << " and > 0. \n";
  cerr << "What is the radius of arc " << j+1 << "? \t";
  cin >> arcRadius;
}
      cerr << "What is its angle?";
      cin >> arcAngle;
      start = Add_Arc(start, arcRadius, M_PI*arcAngle);
    }
  return start;
} //end Node *GetInitialData


void Simulate (Node *start, int &Runs, int iter, int iter2, int iter3)
{
  double theta;
  Node *current, *previous;
  for (current = start, previous = NULL; current != NULL;
       previous = current, current = current->next)
    {
      if (current->arc.distance == 20)
{
  current->arc.angle = ((double)iter/(double)MAX_ITER)*M_PI;
  current->arc.numberOfBoundaryHits = 0;
  //cerr << current->arc.angle << "\n";
}
      else if (current->arc.distance == 50)
{
  current->arc.angle = ((double)iter2/(double)MAX_ITER)*M_PI;
  current->arc.numberOfBoundaryHits = 0;
  //cerr << current->arc.angle << "\n";
}
      else if (current->arc.distance == 35)
{
  current->arc.angle = ((double)iter3/(double)MAX_ITER)*M_PI;
  current->arc.numberOfBoundaryHits = 0;
  //cerr << current->arc.angle << "\n";
}
      else
{
```

```
  current->arc.numberOfBoundaryHits = 0;
}
    }
  for (int i = 0; i < Runs; i++)
    {
      double coord[2][2] = {{0,0}, {0,0}};
      for (int j = 1; (j <= 10000000) &&
     (CheckWholeBoundary(start, coord) == 0); j++)
{
  theta = PickTheta();
  Move(coord, theta);
}
      if (i % 1000 == 0) cerr << "*";
    }
  cerr << "\n";
} // end Simulate
```