AN OVERVIEW OF
GREEN’S FUNCTIONS ON METRIZED GRAPHS

DOUGLAS LI

ABSTRACT. The basic objects of this study are “metrized graphs” and real- or complex-valued functions on such graphs. In particular, we deal with a Laplacian operator on these graphs and the corresponding “Green’s functions” used to invert it. Metrized graphs serve as an accessible means to model general theories (namely those of spectral theory and singularity theory) in higher dimensions, but also have many unique and interesting features in their own right.

CONTENTS

Introduction 1
1. Metrized Graphs 2
2. Laplacian on the space $\mathcal{CPA}(\Gamma)$ 4
3. Classic Problems in Harmonic Analysis 6
4. Generalize to Larger Spaces $\mathcal{Z}(\Gamma)$ and $\mathcal{D}(\Gamma)$ 7
5. Probabilistic Interpretations 8
6. Conclusion 9
7. Applications to Singularity Theory 10
Appendix 11
References 23

INTRODUCTION

This paper is concerns functions, measures, and the Laplacian operator on metrized graphs. A metrized graph $\Gamma$ is a finite connected graph equipped with parameterizations and weights on each of its edges so that it becomes a metric space. (A more rigorous definition is given in §1.) In particular, $\Gamma$ is a one-dimensional manifold except at finitely many branch points that resemble an $n$-pointed star. In this regard, we can consider graphs as analytic objects rather then purely combinatorial. [BR]

Date: September 3, 2004.
Research conducted under the supervision of Mattias Jonsson at the University of Michigan, May–July, 2004.
Supported by the NSF.
We can perform a Laplacian operation on certain continuous functions that map from $\Gamma$ to $\mathbb{C}$. Intuitively, the Laplacian of some well-behaved continuous $u$ is $-u''$ and is denoted $\Delta u$, but $\Delta u$ can be interpreted as both a function itself or as a measure. (A precise definition is given in §2.) Since the resulting function may no longer be continuous and lie in the same space, it is desirable to interpret $\Delta u$ as some measure, a canonical measure. Thus, we implicitly associate a function (of the appropriate space) to a measure in this manner. Furthermore, if given a measure (of the appropriate space and satisfying certain conditions) and a function value at some fixed point, we might be able to uniquely determine the function that associates to that measure. Functions derived in this way are known as Green’s functions.

Our objective is to characterize Green’s functions associated with non-negative discrete measures, to describe necessary and sufficient conditions in order to uniquely determine the Green’s function associated with a given measure (of the appropriate space), and to formulate (and validate) a procedural method of computing such functions given such measures.

Applications lie in potential theory, Markov chains, singularity theory, spectral theory, arithmetic geometry, and mathematical biology.

The study of the Laplacian on metrized graphs was first introduced by Rumely in [Rum], and subsequently further developed by Chinburg and Rumely in [CR] and by Zhang in [Zh]. Metrized graphs arise naturally in the study of arithmetic geometry, and their Green’s functions are related to pairings that occur in arithmetic capacity theory. [BR]

1. Metrized Graphs

A rigorous definition of a metrized graph is best explained in [BR] and is reproduced here for convenience.

**Definition 1.1 (Metrized Graph).** A metrized graph $\Gamma$ is a (non-trivial) compact connected metric space such that $\forall p \in \Gamma$, $\exists$ a radius $\varepsilon_p \in \mathbb{R}^+$ and $\exists n_p \in \mathbb{N}$ such that $B(p; \varepsilon_p)$ is isomorphic to $\{ z \in \mathbb{C} : z = te^{k\cdot 2\pi i / n_p} \text{ for } 0 \leq t < \varepsilon_p \text{ and } k \in \mathbb{Z} \}$.

Denote the metric on $\Gamma$ as $d_\Gamma$.

Note that for each point $p \in \Gamma$, $n_p$ must be unique, or else there would be an obvious contradiction. Furthermore, this definition does not distinguish between vertices and edges as in the usual (combinatorial) notion of a (simplicial) graph since it is not strictly necessary to do so. However, it is convenient to adopt the following convention.

**Definition 1.2.** We call $x \in \Gamma$ a vertex of $\Gamma$ if $n_p \neq 2$. Hence we refer to the set of all vertices of $\Gamma$ as the vertex set $V_\Gamma$.

**Definition 1.3.** We call $e \subseteq \Gamma$ an edge of $\Gamma$ if it is a connected component of $\Gamma$ that does not contain any vertices, and denote the set of all edges of $\Gamma$ by $E_\Gamma$. 

In addition, each edge $e_k$ has an associated length

$$L(e_k) := \sup_\gamma \int_0^1 |\gamma'(t)| \, dt,$$

where the sup is taken over all simple, smooth paths $\gamma : [0, 1] \to e_k$. (This value is not necessary the distance between the endpoints, but must be positive.) Also, we may denote this by $L_{ij}$ and $L(e_{ij})$, where we are referring to the edge adjoining $p_i, p_j \in \Gamma$, provided that there is such an edge and it is unique. The weight of an edge $e_k$ is $w(e_k) := 1/L(e_k)$. Moreover, each edge $e_k$ is thus homeomorphic to the open interval $(0, L_k) \subseteq \mathbb{R}$ or to $\partial \mathbb{D}$. (Observe that it must be the case that $|\Gamma| \geq 0 \wedge |E(\Gamma)| \geq 1 \forall \Gamma$.)

1.1. Relation to Finite Weighted Graphs. These metrized graphs and finite weighted graphs are closely related. It can be shown that there is a bijective correspondence between metrized graphs and equivalence classes of finite weighted graphs [BR] (those connected and having non-degenerate edge set) that is consistent with the definitions of vertex and edge given above by extending the vertex set by a finite number of points as to avoid self-loops and multiple edges. Note that in our definition of metrized graphs we do not allow an isolated point or empty set since these are somehow unrelated to one-dimensional manifolds.

For a given $\Gamma$, let $G$ be some graph in the equivalence class of finite weighted graphs corresponding to $\Gamma$ whose set of vertices is $\tilde{V}_\Gamma = \{v_1, \ldots, v_n\}$. By this construction, if there exists and edge between $v_i$ and $v_j$ then it is unique and there are no edges from $v_k$ to $v_k$. In §2 and §4, we introduce spaces of functions on $\Gamma$. If we equip a function $f \in CPA(\Gamma)$ (resp., $f \in Z\mathcal{H}(\Gamma)$) to $\Gamma$ then let $\tilde{V}_\Gamma(f)$ denote $\tilde{V}_\Gamma$ union with the finite set of points that do not satisfy the affine (resp., $C^2$) condition for $f$. Let $\mathbf{f}$ denote the column vector produced by the values of $f$ at each point corresponding to the elements of $\tilde{V}_\Gamma(f)$ in consistent order. Let $A$ be the symmetric weighted adjacency matrix $D$ be the diagonal weighted degree matrix both associated with $G$ and consistent with the ordering of $\tilde{V}_\Gamma$ (or $\tilde{V}_\Gamma(f)$ if a function is equipped). Note that if there is no edge between $v_i$ and $v_j$ then $A_{ij}=0$, otherwise $A_{ij}=w_{ij}$, where $p_k \in \Gamma$ is associated with $v_k \in G$.

1.2. Metrized Graphs with Ground Nodes. In certain situations, for instance circuit theory and singularity theory, it is sometimes useful to distinguish one or more points $x_0$ as so-called ground nodes, where we have a fixed function value of $f(x_0)=0$. We call a metrized graph $\Gamma$, a metrized graph with ground node if $x_0 \in \Gamma$ is the unique ground node in $\Gamma$, and let $\Gamma'$ denote the part of $\Gamma$ excluding $x_0$ and any edges that have $x_0$ in their closure. We can also consider metrized graphs with multiple (yet finite) ground nodes, but due to the nature of the problems that are under consideration it is possible wlog to collapse or contract all ground nodes into one for purposes of computations. We implicitly include $x_0$ as a vertex of $\Gamma$. First, we must define what it means to take the Laplacian of a function and see how to account for the Laplacian at multiple ground nodes due to such contraction.
Let us consider the space of continuous, piecewise-affine (i.e., piecewise-linear), complex-valued (or real-valued depending on the context) functions on $\Gamma$, denoted $\mathcal{CPA}(\Gamma)$. Note that for such a function to be defined on $\Gamma$, it suffices to define the function values at each of the vertices. For each point $p \in \Gamma$, denote the set of the $n_p$ “formal unit vectors emanating from $p$” by $\text{Vec}(p)$.

For $\vec{v} \in \text{Vec}(p)$, the direction derivative (or one-sided derivative)

$$d_{\vec{v}}f(p) := \lim_{t \to 0^+} \frac{f(p + t\vec{v}) - f(p)}{t}.$$ 

The atomic mass of $f \in \mathcal{CPA}(\Gamma)$ at a point $p$, is given by

$$m_f(p) := -\sum_{\vec{v} \in \text{Vec}(p)} d_{\vec{v}}f(p).$$

We adopt the convention of having the negative sign. It is easy to see that if $p$ is on an edge and $f$ is affine at $p$, then $m_f(p)=0$, which agrees with the usual notion of Laplacian on $\mathbb{R}$ where the function is linear. In the case of $\mathcal{CPA}(\Gamma)$ we say that $f$ is harmonic at $p$ iff $m_f(p)=0$. This is equivalent ([LP]) to the “weighted average” statement that

$$f(p) = \frac{\sum_{p \sim q} f(q) \, w(p, q)}{\sum_{p \sim q} w(p, q)},$$

where $p \sim q$ indicates that $p, q$ are adjacent in $\Gamma$.

For $f \in \mathcal{CPA}(\Gamma)$, we define the Laplacian $\Delta f$ of $f$ to be the discrete measure

$$\Delta f := \sum_{p \in \Gamma} m_f(p) \cdot \delta_p,$$

where $\delta_p$ is the discrete probability measure on $\Gamma$ with the property $\int_{\Gamma} h(x) \, \delta_p(x) = h(p), \, \forall h \in \mathcal{C}(\Gamma)$. As an immediate consequence of this definition, we have the following that shall serve as a necessary condition for discrete measures in solving for associated Green’s functions.

**Lemma 2.1 (Cancellation Property).** For $f \in \mathcal{CPA}(\Gamma)$, let $\mu = \Delta f$. We have that $\int_{\Gamma} d\mu(x) = 0$. 


Proof. As detailed in 1.1, let \( \tilde{V}_\Gamma(f) = \{v_1, \ldots, v_n\} \) and \( p_i \in \Gamma \) be associated to \( v_i \in G \).

\[
\int_\Gamma d\mu(x) = \sum_{i=1}^n m_f(p_i)
\]

\[
= \sum_{i=1}^n \left[ - \sum_{\vec{v} \in \text{Vec}(p_i)} d_{\vec{v}} f(p_i) \right]
\]

\[
= \sum_{i=1}^n \sum_{j=1}^n A_{ij} (f(p_i) - f(p_j))
\]

\[
= \sum_{i=1}^n \sum_{j=1}^n A_{ij} f(p_i) - \sum_{i=1}^n \sum_{j=1}^n A_{ji} f(p_i)
\]

\[
= \sum_{i=1}^n \sum_{j=1}^n A_{ij} f(p_i) - \sum_{i=1}^n \sum_{j=1}^n A_{ij} f(p_i)
\]

\[
= 0
\]

by symmetry of \( A \).

This definition of \( \Delta \) on \( CPA(\Gamma) \) is essentially equivalent to the classical definition of the Laplacian matrix \( Q \) associated to a finite weighted graph. In fact, we can make a stronger statement in the case of \( CPA(\Gamma) \).

**Lemma 2.2.** Fix \( f \in CPA(\Gamma) \). Take \( Q \) to be the \( n \times n \) Laplacian matrix of \( G \) whose set of vertices is \( \tilde{V}_\Gamma(f) = \{v_1, \ldots, v_n\} \), where we incorporate the finite set of points that do not satisfy the affine condition for \( f \). (Recall that \( Q := D - A \), where \( D \) and \( A \) were explained above.) Then we have that \( [\Delta f] = Qf \), where the \( i \)th entry of the column vector \( [\Delta f] \) is \( c_i \), taken from \( \Delta f = \sum_{i=1}^n c_i \delta_{p_i} \).

**Proof.** We have

\[
[\Delta f]_i = m_f(p_i)
\]

\[
= - \sum_{\vec{v} \in \text{Vec}(p_i)} d_{\vec{v}} f(p_i)
\]

\[
= \sum_{j=1}^n A_{ij} (f(p_i) - f(p_j))
\]

\[
= \deg(p_i) f(p_i) - \sum_{j=1}^n A_{ij} f(p_j)
\]

\[
= D f - A f
\]

\[
= (Qf)_i
\]

for \( i = 1, \ldots, n \).
Once generalized further, the Laplacian on a metrized graph is a non-trivial yet computationally accessible variant of the Laplacian on a higher-dimensional Riemannian manifold.

3. Classic Problems in Harmonic Analysis

Given a discrete measure on \( \Gamma \) we would like to reconstruct the corresponding function \( f \in \mathcal{CPA}(\Gamma) \). (As we will show, this function is unique up to a constant.) As in complex analysis, harmonic functions we consider, satisfy a maximum principle. For \( X \subseteq \overline{V}_\Gamma \), let \( \overline{X} \) denote the set of vertices either in \( X \) or adjacent to \( X \).

**Theorem 3.1** (Maximum Principle). Let \( X \) be a connected subset of \( V_\Gamma \), \( f \in \mathcal{CPA}(\Gamma) \), \( f(\Gamma) \subseteq \mathbb{R} \), \( f \) harmonic on \( X \). If \( f \) attains a maximum on \( \Gamma \) in \( X \), then \( f \upharpoonright X = \max f \).

**Proof.** Let \( K := \{ p \in \overline{X} : f(p) = \max f \} \). Observe that if \( p \in X \cap K \) and \( p \sim q \) then \( f(q) = \max f \) by harmonicity of \( f \) at \( p \). Thus, \( K \cap \overline{K} = K \). Since \( H \) is connected and \( K \neq \emptyset \), it follows that \( K = \overline{X} \). \( \square \)

We now examine the Metrized graph analog of the Dirichlet problem.

**Theorem 3.2.** Fix \( \Gamma \) and let \( X \) be a proper subset of \( V_\Gamma \). There exists a unique solution \( u \in \mathcal{CPA}(\Gamma) \) such that

\[
\begin{align*}
\Delta u(p) &= 0 \quad \text{for } p \notin X \\
u &= \text{given on } X
\end{align*}
\]

**Proof.**

**Existence:** It can easily be verified that \( u(p) = E_p[u(q)] \) is such a solution, where \( q \) is a random variable for the first vertex in \( X \) visited by a (discrete) network random walk starting at \( p \). This is because \( u \) on \( X \) can now be written as a weighted average of neighboring function values.

**Uniqueness:** Suppose \( u_1, u_2 \) both satisfy the conditions above. Let \( f = u_1 - u_2 \). It suffice to show \( f \leq 0 \). Note that \( f = 0 \) on \( X \) so we suppose \( f > 0 \) on some “edge-connected” component \( K \) of \( V_\Gamma \setminus X \). From the Maximum principle, \( f \) is constant on \( K \), and this constant must be 0. This contradiction shows that \( f \leq 0 \), by a similar argument \( f \geq 0 \), and so \( f = 0 \) as desired. \( \square \)

**Corollary 3.3.** If \( \Delta u(p) = 0 \) for all \( p \in V_\Gamma \), then \( u \) is a constant function.

Similarly, we have the Poisson problem.

**Theorem 3.4.** Given \( X \) a proper subset of \( V_\Gamma \) and \( u_1, \ldots, u_n, c_1, \ldots, c_m \in \mathbb{R} \). There exists a unique solution \( u \in \mathcal{CPA}(\Gamma) \) for

\[
\begin{align*}
(\Delta u)(y_i) &= c_i, \quad \text{for } V_\Gamma \setminus X = \{y_1, \ldots, y_m\} \\
u(x_j) &= u_j, \quad \text{for } X = \{x_1, \ldots, x_n\}
\end{align*}
\]

**Proof.** We know that \( \Delta \) is a linear operator on \( V_\Gamma \). Applying Lemma 2.2, let \( A = \begin{bmatrix} Q & 0 \\ 0 & I_n \end{bmatrix} \in M_{m+n}(\mathbb{R}) \), where \( Q \) is the \( m \times m \) Laplacian matrix introduced previously. To show existence and uniqueness for the system given above, it is
enough to show that dim(Im(A)) = m + n or equivalently dim(Ker(A)) = 0. In turn, it now suffices to simply show uniqueness of \( f \) in the linear equation \( A \cdot f = 0 \), but this follows immediately from the same argument given in the proof of Theorem 3.2 using the Maximum principle.

**Theorem 3.5.** For \( V_\Gamma = \{p_1, \ldots, p_n\} \), let \( c_1, \ldots, c_n \in \mathbb{R} \) such that \( \sum_{i=1}^{n} c_i = 0 \). There exists a solution \( u \in \mathcal{CPA}(\Gamma) \) to \( (\Delta u)(p_i) = c_i \) for \( i = 1, \ldots, n \).

**Proof.** First note that it was an established condition in Lemma 2.1 that \( \sum_{i=1}^{n} c_i = 0 \) for there to exists a solution. If there exists a solution, it is certainly not unique since we can shift by a constant. However, uniqueness given a the function value at some fixed point is guaranteed by the Maximum principle. It remains to show that \( \sum_{i=1}^{n} c_i = 0 \) is a sufficient condition. Since \( \Delta \) is a linear operator it suffices to show that dim(Ker(\( \Delta \))) = 1 and thus dim(Im(\( \Delta \))) = n - 1.

\[
\text{Ker}(\Delta) = \{ f \in \mathcal{CPA}(\Gamma) : (\Delta f)(p) = 0 \ \forall p \in V_\Gamma \} \\
= \{ f \in \mathcal{CPA}(\Gamma) : f \text{ is constant} \} \\
\simeq \mathbb{R}
\]

so dim(Ker(\( \Delta \))) = 1 as desired.

Observe that all Theorems given in this section can also be restated in terms of complex-valued functions. This is because we can use the Maximum principle on both the real and imaginary parts of a function and the proofs proceed in a similar fashion.

### 4. Generalize to Larger Spaces \( \mathcal{Z}_h(\Gamma) \) and \( \mathcal{D}(\Gamma) \)

So far we have restricted to looking at discrete measures. To consider other (bounded, signed) measures we must generalize to a larger space of functions; One can consider define a more general class of functions other then \( \mathcal{CPA}(\Gamma) \). We define the Zhang space \( \mathcal{Z}_h(\Gamma) \) to be the set of all functions \( f : \Gamma \rightarrow \mathbb{C} \) such that \( f \) is piecewise \( C^2 \) and \( f''(x) \in L^1(\Gamma) \) [Zh]. The resulting Laplacian operator combines aspects of the discrete Laplacian previously described and the continuous Laplacian \(-f''(x)dx\) on \( \mathbb{R} \), where \( dx \) is the Lebesque measure.

**Definition 4.1.** The Laplacian of a function \( f \in \mathcal{Z}_h(\Gamma) \) is given by the measure

\[
\Delta f = -f''(x)dx - \sum_{p \in \Gamma} m_f(p) \cdot \delta_p,
\]

where \( \delta_p \) is the Dirac measure (unit point mass) at \( p \) as before. Observe that this new definition agrees with our previous definition for \( \mathcal{CPA}(\Gamma) \), and that the summation is always finite.

Let the class of all function on \( \Gamma \) whose one-sided derivatives exist, for all points and unit vectors of \( \Gamma \), be

\[
\mathcal{D}(\Gamma) = \{ f : \Gamma \rightarrow \mathbb{C} : d_{\vec{v}}f(p) \text{ exists } \forall p \in \Gamma, \vec{v} \in \text{Vec}(p) \}.
\]

Here we describe the Laplacian as follows.
Definition 4.2. Given a set $S$ in $\mathcal{A}$, the Boolean algebra of subsets of $\Gamma$ generated by connected open sets, $S$ is a finite disjoint union of sets isometric to open, half-open, or closed intervals. For $f \in D(\Gamma)$, the measure or mass of such a set is given by

$$m_f(S) = \sum_{p \in \partial S} \sum_{\bar{v} \in \text{In}(p,S)} d_{\bar{v}} f(p) - \sum_{p \in \partial S} \sum_{\bar{v} \in \text{Out}(p,S)} d_{\bar{v}} f(p),$$

where $\text{In}(p,S)$ denotes the set of inward-directed unit vectors at $p$ and $\text{Out}(p,S) = \text{Vec}(p) \setminus \text{In}(p,S)$.

In this way, $m_f$ defines a finitely additive set function on $\mathcal{A}$ such that $m_f(\emptyset) = m_f(\Gamma) = 0$ and $m_f\{p\} = m_f(p)$. It also follows that $m_f(\Gamma \setminus S) = -m_f(S)$ and $m_{af+bg} = am_f + bm_g$. For a detailed verification of these statements, please refer to [BR].

It can be shown that $\mathcal{CPA}(\Gamma) \subset Z_h(\Gamma) \subset D(\Gamma) \subset C(\Gamma)$, and that each of these definitions agree when restricted lower subsets of the space. Thus, it suffice to use only the last definition, but the progression shows the motivation for such a definition.

5. Probabilistic Interpretations

The topic discussed in this paper is closely related to the study of transience and recurrence of irreducible Markov chains. In fact we are only interested in reversible Markov chains, where we call a Markov chain reversible if there is a positive function $x \mapsto \pi(x)$ on the state space such that the transition probabilities satisfy $\pi(x)p_{xy} = \pi(y)p_{yx}$ for states $x, y$. We make a graph by taking the states of the Markov chain for vertices and joining vertices identified with $x, y$ by an edge when $p_{xy} > 0$, assigning a weight of $c(x,y) := \pi(x)p_{xy}$ to the edge. With this graph, the Markov chain may be described as a network random walk. Conversely, every weighted, connected graph such that the sum of the weights incident to each vertex is finite describes a random walk with transition probabilities proportional to the weights. Moreover, such a random walk is an irreducible reversible Markov chain by simply defining $\pi(x)$ to be the sum of the weights incident to $x$.

5.1. Electric Networks and Random Walks. In order to study certain problems in harmonic analysis, especially when sequences of subgraphs of an infinite graph are involved, it is useful to consider electrical networks. Since they have physical meaning, electrical networks provide intuition, yet can be made as rigorous as necessary.

Mathematically, an electric network is simply a weighted graph, where the weights of the edges are called conductances; the lengths of the edges are the reciprocals or resistances. The function values are analogous to the potentials or voltages, and the Laplacian is related to the amount of current. In this way the common circuit analysis laws can be restated in the language of graphs or metrized graphs. Conversely, we can apply the tools from circuit analysis to our
study, such as, network reductions, effective resistance, etc.

In the original consideration of $\mathcal{CPA}(\Gamma)$, we can simply use the most basic circuit analysis tools to solve for the potentials given the measure, which is no surprise since the problems in $\mathcal{CPA}(\Gamma)$ are essentially linear algebra problems. However the study of higher spaces of functions is incomplete, and would be analogous to allowing for the flow of current into or out of a wire to be a distribution (since each point is of measure zero, no current flows through any point, yet there is current going into or out of the wire).

Although various authors mention that it is possible to solve for such functions in this more general case, there does not appear to be a methodical, way of doing so in the literature. Thus I wrote some code to both solve and generate such measures. I used Mathematica and specifically the Combinatorica package. The code is provided in the Appendix.

6. Conclusion

Given any signed Borel measure on a metrized graph $\Gamma$, do we necessarily have existence and uniqueness of a solution in $\mathcal{D}(\Gamma)$? An outline of a proof of existence and uniqueness follows.

Existence follows from the same argument given previously. Intuitively, we may construct an auxiliary graph and discrete measure by collapsing each of the continuous components into a vertex and appending edges of measure zero to maintain the proper connections. We can then solve this problem in the $\mathcal{CPA}(\Gamma)$ context. The one-sided derivatives determined can be used to solve for the function values at the original edges, by performing standard integrations. Thus, reducing the problem from $\mathcal{D}(\Gamma)$ to one in $\mathcal{CPA}(\Gamma)$ and some finite number in $\mathbb{R}$ context. This uses the concept of network reduction from circuit theory and can be made rigorous.

Although this approach works, a more efficient way of solving for the $2E$ variables (the one-sided derivatives) is by extracting $E$ equations for each edge, $V - 1$ equations for each vertex except the ground node, and $E - V + 1$ equations for each loop (the sum of function value change around a loop should be zero since the function is continuous). This is essentially how the code in the appendix performs the solve.

Thus it follows that there is a bijection between $\mathcal{D}(\Gamma)$ and signed Borel measures on $\Gamma$ where $\Gamma$ has a ground node, and a similar statement for the complex case. Although outside the scope of this paper, it can be shown that this bijective correspondence is in fact homeomorphic.
7. Applications to Singularity Theory

The study of metrized graphs naturally arises in various topics. For example: they are used to study arithmetic intersection theory on algebraic curves in number theory (see [CR], [Zh]), neuron transmission in mathematical biology, and in wave-propagation models in physics.

This research however, arises from the study of a certain Valuative Tree in singularity theory.

7.1. The Valuative Tree. The valuative tree $V$ is the set of all $\mathbb{R} \cup \{\infty\}$ valued valuations $\nu$ on $\mathbb{C}[x,y]$ normalized by $\min\{\nu(x), \nu(y)\} = 1$ ([FJ]) with the natural structure of an $\mathbb{R}$-tree, induced by the order relation $\nu_1 \leq \nu_2$ iff $\nu_1(\phi) \leq \nu_2(\phi)$ for all $\phi$ [FJ]. It can also be metrized and viewed as a metrized graph, or more specifically metrized tree [FJ]. Algebraically, this structure can be obtained by taking a suitable quotient of the Riemann-Zariski variety of $\mathbb{C}[x,y]$, forcing it to be a Hausdorff topology [FJ]. There are several other means of arriving at a structure isomorphic to $V$, and these are explained in [FJ].
Here is the code used to solve and generate Greens functions considered in this paper. It is split into five sub-sections: giving routine and startup commands, a solver for the simple/nodal case, a generator for the nodal case, a solver for the more complicated case, and a generator for the complicated case. During the testing phase, many bugs were discovered that were an unavoidable part of the Combinatorica package, which I notified the authors of and they may fix in subsequent versions. (These problems were mainly in regards to overlapping text and haphazard support for multi-graphs. It is possible to download the latest version from www.combinatorica.com, even if you are using Mathematica 4.x, which would at least render arced edges whenever appropriate.)

7.2. Routine and Startup Commands.

(* The Combinatorica package is fairly standard to use for graph theory study. The "New" is not necessary inversion 5 of Mathematica, but is in older versions. It would also be necessary to download and place the package to the proper AddOn directory. *)

<< DiscreteMath'NewCombinatorica'

Off[Graphics::"gprim"] (* Just to turn off certain warning messages *)

(* This is a simple function to flip the orientation of a subset of edges on a di-graph while preserving internal edge ordering *)

FlipEdges[g_Graph, L_List] :=
  ChangeEdges[g, Table[If[MemberQ[L, i], Reverse[Edges[g][[i]]], Edges[g][[i]]],
    {i, 1, Length[g]}]];

(* Just a shortcut for something used frequently *)

GetEdgeLength[g_Graph, i_Integer] := GetEdgeWeights[g, Edges[g][[i]]];

(* Given any matrix, this function deletes the last row and last column. *)

SubMatrix[m_List] := Drop[Transpose[Drop[Transpose[m], -1]], -1];

(* Gives a linear extension of a function on the vertex set of a graph to the edges assuming that the edges have measure zero. *)

AffineEdgeFunctions[g_Graph] :=
  Module[{
    v = GetVertexWeights[g],
    e = GetEdgeWeights[g],
    e_Graph = Edges[g],
    Table[If[e_Graph[[i]] == 0, v[[v_Graph[[i]]]], v[[e_Graph[[i]]]] - 
      t * e_Graph[[i]][[3]] - v[[v_Graph[[i]]]]],
    {i, 1, Length[g]}]];;
Given an Undirected graph, this function forces an orientation on the edges in a natural way and returns it as a Directed graph, preserving the graph options, but edge lengths and labelling are not yet reliable.

```
ForceOrientation[g_graph] :=
  ChangeVertices[
    SetVertexWeights[
      SetEdgeWeights[
        SetVertexLabels[
          SetEdgeLabels[
            SetGraphOptions[FromAdjacencyMatrix[
              Table[If[j > i, ToAdjacencyMatrix[g_graph, 0], {i, V[g]}], {j, V[g]}],
              Type -> Directed], 
              SetEdgeLabels[g_graph], GetVertexLabels[g_graph], GetEdgeWeights[g_graph], GetVertexWeights[g_graph]], Vertices[g_graph];

NumberedEdges[g_graph] :=
  SetEdgeLabels[g_graph, Table[ToString[i] <> "", {i, M[g_graph]}];

StylizeGraph1[g_graph] := SetGraphOptions[1,
  VertexColor -> Orange, VertexStyle -> Disk[.05],
  EdgeColor -> LightBlue, EdgeStyle -> Thickness[.012],
  Background -> LightYellow, VertexNumberPosition -> Center,
  EdgeLabelPosition -> LowerRight, VertexLabelPosition -> Center,
  VertexNumberColor -> Green, VertexLabelColor -> Green, EdgeLabelColor -> Purple,
  TextStyle -> {FontFamily -> "Arial", FontWeight -> "Bold", FontSize -> 14}];

StylizeGraph2[g_graph] := SetGraphOptions[2,
  VertexColor -> Red, VertexStyle -> Disk[.07],
  EdgeColor -> Purple, EdgeStyle -> Thickness[.02],
  Background -> LightBlue, VertexNumberPosition -> Center,
  EdgeLabelPosition -> LowerRight, VertexLabelPosition -> Center,
  VertexNumberColor -> White, VertexLabelColor -> White, EdgeLabelColor -> Black,
  TextStyle -> {FontFamily -> "GothicE", FontWeight -> "Bold", FontSize -> 16}];

StylizeGraph3[g_graph] := SetGraphOptions[3,
  VertexColor -> Black, VertexStyle -> Disk[.07],
  EdgeColor -> Yellow, EdgeStyle -> Thickness[.015],
  Background -> Tomato, VertexNumberPosition -> Center,
  EdgeLabelPosition -> LowerRight, VertexLabelPosition -> Center,
  VertexNumberColor -> Yellow, VertexLabelColor -> Yellow, EdgeLabelColor -> Black,
  TextStyle -> {FontFamily -> "Times", FontWeight -> "Bold", FontSize -> 16}];

StylizeGraph4[g_graph] := SetGraphOptions[4,
  VertexColor -> Blue, VertexStyle -> Box[.07],
  EdgeColor -> Gold, EdgeStyle -> Thickness[.015],
  Background -> AliceBlue, VertexNumberPosition -> Center,
  EdgeLabelPosition -> LowerRight, VertexLabelPosition -> Center,
  VertexNumberColor -> Gold, VertexLabelColor -> Gold, EdgeLabelColor -> Blue,
  TextStyle -> {FontFamily -> "Modern", FontWeight -> "Bold", FontSize -> 16}];
```
7.3. Solver for Nodal Case.

```math
q = Graph[{{1, 2}}, {{2, 3}}, {{2, 4}}, {{3, 4}}],
{{0, -1}}, {{0, 0}}, {{-1, 1}}, {{1, 1}}], EdgeDirection -> On];
ShowGraph[q = StylizeGraph[NumberedEdges[q]], VertexNumber -> True,
EdgeLabel -> True]
```

* Graphics *

```math
(* Set Ground Nodes *)
GroundNodes = {1};
If[Range[VertexCount[q]] == GroundNodes, "GroundNodes are OK",
"Error: GroundNodes Invalid!!!"]
```

GroundNodes are OK

```math
q0 = NumberedEdges[Contract[q, GroundNodes]]; 
ShowGraph[q0, VertexNumber -> True, EdgeLabel -> True]
```

* Graphics *

```math
(* Check for Connectedness *)
If[ConnectedQ[q0], "The transformed graph is Connected.",
"Error: Transformed Graph not Connected!!!"]
```

The transformed graph is Connected.
Optional: \( q_0 = \text{SetEdgeWeights}[q_0, \{1, 1, 1, 1\}] \); (Edge lengths default to 1. Array \( M[q_0] \) would set all the edge lengths to 2.)

\( q_1 = \text{SetEdgeLabels}[\text{SetVertexLabels}[q_0, \{"X_1", "X_2", "X_3", "X_4"\}], \text{GetEdgeWeight}[q_0]]; \)

ShowGraph[\( q_1 = \text{SetGraphOptions}[q_1, \text{VertexLabel} \rightarrow \text{True}, \text{EdgeLabel} \rightarrow \text{True}] \)]

- Graphics -

This will serve to hold the point mass values. A point mass may only occur at vertices (the user may extend the graph as necessary). Vertex weights default to 0 as desired. No point mass may be set for the ground node (it will be forced/calculated later). The size of this list must be \( V[q_0] - 1 \). Currently we only allow for NON-NEGATIVE point masses.

\( \text{PointMasses} = \{0, 0, 1\} \);

If \( \{\text{Length}[\text{PointMasses}] + 1 \neq V[q_0]\} \) \& \& \( \text{Table}[\text{PointMasses}[i] \geq 0, \{i, V[q_0] - 1\}] \), "PointMasses are OK", "Error: PointMasses Invalid!!!"

PointMasses are OK

This will serve to hold the edge measure expressions. Ideally we only want to consider \( C^1 \) functions (the user may extend the graph as necessary, but it might still work). The default should be the constant function 0, but for now all edge measures must be explicitly input. The size of this list must be \( M[q_0] \). Currently we only allow for NON-NEGATIVE measures. An edge is parameterize in terms of \( t \) starting from the tail of the directed edge.

\( \text{EdgeMeasures} = \text{FullSimplify}[\{0, 0, 0, 0\}] \);

Here we compute the mass of each edge. This is then verified to be non-negative and integrable (or perhaps we want to allow for numeric approximations for definite integration).

\( \text{Module}[\{t\}, \text{EdgeMeasures} = \text{Table}[\int_{0}^{\text{GetEdgeLength}[q_0, 1]} \text{EdgeMeasures} \, dt, \{i, M[q_0]\}] ]; \)

If \( \{\text{And} \& \& \text{Table}[\text{TrueQ}[\text{Simplify}[x \geq 0, 0 \leq x \leq \text{GetEdgeLength}[q_0, 1]], \{i, M[q_0]\}] \} \), "Edge Measures are OK", "Error: Edge Measures Invalid!!!"

Edge Measures are OK

\( \text{AppendTo}[\text{PointMasses}, -\{\text{Plus} \& \& \text{PointMasses} + \text{Plus} \& \& \text{EdgeMeasures}\}] \);

\( \text{Clear}[t]; \text{EdgeIntegrals} = -\int \text{EdgeMeasures} \, dt; \)

If \( \text{Moda} = \{\text{EdgeMeasures} == \text{Table}[0, \{M[q_0]\}]\}, "This is the simple case.", "This is the more complicated case." \)

This is the simple case.
Module[{ModalMat = Table[0, {i, V[g0]}], {j, V[g0]}]}, t, len],
For[i = 1, i ≤ M[g0],
len = If[GetEdgeWeights[g0][i][j][k] = 0, t, GetEdgeWeights[g0][i][j][k]^{-1}];
ModalMat[[Edges[g0][i][j][k], Edges[g0][i][j][k][k]]] += len;
ModalMat[[Edges[g0][i][j][k], Edges[g0][i][j][k][k]]] -= len;
ModalMat[[Edges[g0][i][j][k], Edges[g0][i][j][k][k]]] += len; i++];
g2 = SetVertexWeights[g0, Append[Limit[LinearSolve[SubMatrix[ModalMat], Drop[PointMasses, -1]], t = 0], 0]];
EdgeFunctions = AffineEdgeFunctions[g2];
]
g2 = SetVertexLabels[g2, Table["u = " <> ToString[GetVertexWeights[g2][i]] <> "", i, V[g2]]];
g2 = SetEdgeLabels[g2, Table["u = " <> ToString[EdgeFunctions[g2][i]] <> ", A = 0" <> "", i, M[g2]]];
G2 = SetGraphOptions[g2, VertexLabel -> True, EdgeLabel -> True, VertexLabelPosition -> LowerRight, VertexLabelColor -> Red, TextStyle -> {FontSize -> 10}];
ShowGraphArray[{g1, g2}, PlotRegion -> {{0, .9}, {0, 1}}]
7.4. Generator for Nodal Case.

\( g = \text{Graph}([[1, 2]], [[2, 3]], [[3, 4]], [[4, 5]]), \\
\text{EdgeDirection} \rightarrow \text{on}; \\
\text{ShowGraph}[g = \text{StylizedGraph}[\text{NumberedEdges}[g]], \text{VertexNumber} \rightarrow \text{True}, \\
\text{EdgeLabel} \rightarrow \text{True}]; \\
\text{SetGroundNodes} \rightarrow \text{GroundNodes} = \{1\}; \\
\text{If}[\text{Range}[V[g]] \neq \text{GroundNodes}, \text{"GroundNodes are OK"}, \\
\text{"Error: GroundNodes Invalid!!!"}]; \\
\text{GroundNodes are OK} \\
\text{g0} = \text{NumberedEdges}[\text{Contract}[g, \text{GroundNodes}]]; \\
\text{ShowGraph}[g0, \text{VertexNumber} \rightarrow \text{True}, \text{EdgeLabel} \rightarrow \text{True}]; \\
\text{Check for Connectedness} \rightarrow \text{If}[\text{Connected}[g0], \text{"The transformed graph is Connected."}, \\
\text{"Error: Transformed Graph not Connected!!!"}]; \\
\text{The transformed graph is Connected.} \\
\text{Optional} \rightarrow \text{g0} = \text{SetEdgeWeights}[g0, \{1, 1, 1, 1\}]; \\
\text{Edge lengths default to 1. \text{Array}[2, \text{g0}]} \text{would set all the edge lengths to 2.} \\
\text{gl} = \text{SetEdgeLabels}[\text{SetVertexLabels}[g0, \{X1, X2, X3, X4\}], \text{GetEdgeWeights}[g0]]; \\
\text{ShowGraph}[gl = \text{SetGraphOptions}[gl, \text{VertexLabel} \rightarrow \text{True}, \text{EdgeLabel} \rightarrow \text{True}]]; \\
\text{In case user wants to review internal Vertex Numbering for the subsequent step} \rightarrow \\
\text{ShowGraph}[g0, \text{VertexNumber} \rightarrow \text{True}, \text{EdgeLabel} \rightarrow \text{True}]; \\
\text{This will serve to hold the function values on vertices. The function will then be extended linearly to the} \\
\text{edges and thus no non-differential points (corners) may appear in the edges. It is up to the user to} \\
\text{extend the graph as desired. Currently we only allow for NON-NEGATIVE functions, and it is forced} \\
\text{that the function value at the ground node is zero,} \\
\text{although the user has freedom to change this here). The default value is 0 so it is ok to not verify} \\
\text{correct amount of user inputs.} \\
\text{g0} = \text{SetVertexWeights}[g0, \text{Append}[\{1, 2, 3\}, 0 \text{ (value at Ground Node)}]]; \\
\text{If} [\text{And} @ (g > 0 & \# \text{GetVertexWeights}[g0]), \text{"Function Values on Vertices are OK"}, \\
\text{"Error: Function Values Invalid!!!"}]; \\
\text{Function Values on Vertices are OK} \\
\text{Module}[\{\text{NodalMat} = \text{Table}[0, \{i, V[g0]\}, \{j, V[g0]\}], t, \text{len}\}, \\
\text{For}[i = 1, i \leq \text{M}[g0], \\
\text{len} = \text{If}[\text{GetEdgeWeights}[g0][i][j] = 0, t, \text{GetEdgeWeights}[g0][i][j]^{-1}]; \\
\text{NodalMat}[\text{Edges}[g0][i][j], \text{Edges}[g0][i][j]] += \text{len}; \\
\text{NodalMat}[\text{Edges}[g0][i][j], \text{Edges}[g0][i][j]] -= \text{len}; \\
\text{NodalMat}[\text{Edges}[g0][i][j], \text{Edges}[g0][i][j]] -= \text{len}; \\
\text{NodalMat}[\text{Edges}[g0][i][j], \text{Edges}[g0][i][j]] += \text{len}; \\
\text{PointMasses} = \text{NodalMat}.\text{GetVertexWeights}[g0]; \\
\text{EdgeFunctions} = \text{AffineEdgeFunctions}[g2 = g0];]
\texttt{g2 = SetVertexLabels[g2,}
\texttt{ Table["u" \texttt{\rightarrow} \texttt{ToString[GetVertexWeights[g2][u]], StandardForm \texttt{\rightarrow} ", \texttt{\Delta=} \texttt{\rightarrow} \
\texttt{ToString[PointMasses[g2][u] \texttt{\rightarrow}} ", \
\texttt{V[g2]]}, \texttt{]}];}
\texttt{g2 = SetEdgeLabels[g2,}
\texttt{ Table["u" \texttt{\rightarrow} \texttt{ToString[EdgeFunctions[g2][u], StandardForm \texttt{\rightarrow} ", \texttt{\Delta=} \texttt{\rightarrow} \
\texttt{0} \texttt{\rightarrow} ", \
\texttt{V[g2]]}, \texttt{]}];}
\texttt{g2 = SetGraphOptions[g2, VertexLabel \texttt{\rightarrow} \texttt{True, EdgeLabel \texttt{\rightarrow} \texttt{True,}}}
\texttt{VertexLabelPosition \texttt{\rightarrow} \texttt{LowerRight, VertexLabelColor \texttt{\rightarrow} \texttt{Red,}}}
\texttt{TextStyle \texttt{\rightarrow} \texttt{FontSize \texttt{\rightarrow} 12]};}
\texttt{ShowGraphArray[[g1, g2], PlotRegion \texttt{\rightarrow} \texttt{[[0, 0.9], \{0, 1\}]]}
7.5. Solver for General Case.

\[ g = \text{Graph}[\{\{1, 2\}, \{2, 3\}, \{\{0, 1\}, \{1, 2\}, \{2, 11/8\}\}], \]
\[ \text{EdgeDirection -> On}; \]
\[ \text{ShowGraph}[g = \text{StylizeGraph}[\text{NumberedEdges}[g]], \text{VertexNumber -> True,} \]
\[ \text{EdgeLabel -> True}]; \]

(* Set Ground Nodes *)
\[ \text{GroundNodes} = \{1\}; \]
\[ \text{If}[\text{Range}[V[g]] \subset \text{GroundNodes}, \text{"GroundNodes are OK",} \]
\[ \text{"Error: GroundNodes Invalid!!!"}]; \]

GroundNodes are OK

\[ g0 = \text{NumberedEdges}[\text{Contract}[g, \text{GroundNodes}]]; \]
\[ \text{ShowGraph}[g0, \text{VertexNumber -> True, EdgeLabel -> True}]; \]

(* Check for Connectedness *)
\[ \text{If}[\text{Connected}[g0], \text{"The transformed graph is Connected.",} \]
\[ \text{"Error: Transformed Graph not Connected!!!"}]; \]

The transformed graph is Connected.

(* Optional *)
\[ g0 = \text{SetEdgeWeights}[g0, \{1, 1\}]; \]
\[ \text{Edge lengths default to 1. Array}[2, M[g0]] \text{ would set all the edge lengths to 2.} \]
\[ g1 = \text{SetEdgeLabels}[\text{SetVertexLabels}[g0, \{"x_1", \"x_2", \"x_3"\}], \text{GetEdgeWeights}[g0]]; \]
\[ \text{ShowGraph}[g1 = \text{SetGraphOptions}[g1, \text{VertexLabel -> True, EdgeLabel -> True}]; \]

(* This will serve to hold the point mass values. A point mass may only occur at vertices
the user may extend the graph as necessary. Vertex weights default to 0 as desired. No point mass
may be set for the ground node (it will be forced/calculated later). The size of this list must be V[g0] -
1. Currently we only allow for NON-NEGATIVE point masses. *)
\[ \text{PointMasses} = \{1/2, 1/4\}; \]
\[ \text{If}[\{\text{Length}[\text{PointMasses}] + 1 == V[g0]\} \land \text{And} @\text{Table}[\text{PointMasses}_{i^2} \geq 0, \{i, V[g0] - 1\}], \]
\[ \text{"PointMasses are OK","Error: PointMasses Invalid!!!"}]; \]

PointMasses are OK
EdgeMeasures = FullSimplify[{0, 1/4}];
(* Here we compute the mass of each edge. This is then verified to be non-negative and integrable. Or perhaps we want to allow for numerical approximations for definite integration. *)
Clear[t]; EdgeMasses = Table[ integral Edgemesasures_{i} dt, {i, M[0]} ];
If[ And @@ Table[ TrueQ[ Simplify[ x > 0, 0 <= x <= GetEdgeLength[0, 0] ] ], {i, M[0]} ]
 And @@ Table[ TrueQ[ EdgeMasses_{i} >= 0, {i, M[0]} ], "Error: Edge Measures Invalid!!!" ]

Edge Measures are OK

AppendTo[PointMasses, -(Plus @@ PointMasses + Plus @@ EdgeMasses)];
Clear[t]; EdgeDerivatives = - integral EdgeMeasures dt;
If[ Modal == ( EdgeMeasures == Table[0, {i, M[0]} ] ), "This is the simple case.",
 "This is the more complicated case."

This is the more complicated case.

Module[ [ EdgeMat = Table[If[2 i == j || 2 i == j + 1, 1, 0], {i, M[0]}, {j, 2 * M[0]} ];
 VertexMat = Table[ Flatten[ Table[ If[ Edges[0][[i]] == j, 1, 0 ] & /@ {1, 2}, {j, M[0]} ] ];
 {i, V[0] - 1} ], SignMat, L = M[0] - V[0] + 1, thisCycle, tempG = q0, t,
 ChangeInFunction =
 Table[ integral Edgemesasures_{i} dt - ( EdgeDerivatives_{i} / t - 0 ),
 {i, M[0]} ] ];
RHS = Join[ EdgeMasses, -Drop[ PointMasses, -1] ];
FullMat = Join[ EdgeMat, VertexMat ];
For[ i = 1, i <= L, thisCycle = FindCycle[ MakeUndirected[ tempG ] ];
 SignMat = Table[ 0, {i, M[0]} ];
 For[ j = 1, j <= M[0],
 If[ Extract[ thisCycle, {i}, {j} ] ] == Edges[0][[j]], SignMat[[j]] = 1;
 If[ Extract[ thisCycle, {i}, {j} ] ] == Edges[0][[j]], SignMat[[j]] = -1, j += 1, i++ ];
AppendTo[ FullMat, Table[ If[ OddQ[ i ], SignMat[ ] / 2 ], 0 ], {j, 2 * M[0]} ] ];
M[0] AppendTo[ RHS, - Sum[ { SignMat[[i]] * ChangeInFunction } , i = 1 ];
 tempG = DeleteEdge[ tempG, Take[ thisCycle, 2 ] ];
c++ ]; DirectionalDerivatives = LinearSolve[ FullMat, RHS ]; ]}
EdgeFunctions =
  Table[∫ EdgeDerivatives[u] dt - ∫ EdgeDerivatives[u] dt / t = 0] +
  (DirectionalDerivatives[u] - (EdgeDerivatives[u] / t = 0)) t, {i, M[g0]}];
Module[{EdgesDone = Table[False, {i, M[g0]}]}, VerticesDone = Table[False, {i, V[g0]}]],
  edgeList = Edges[g0], Shifts = Table[0, {i, M[g0]}],
  lens = GetEdgeWeights[g, Edges[g]],
  g2 = SetVertexWeights[g0, {V[g0]}, {0}], VerticesDone = True;
While[1 (And && EdgesDone && VerticesDone),
  For[i = 1, i ≤ M[g0],
    If[VerticesDone[edgeList, {i}] && EdgesDone[g, {i}],
      Shifts[{{g2}} + (EdgeFunctions[{{g2}} / t = lens[{{g2}}]]; VerticesDone[{{edgeList, {i}}}] = True,
      If[VerticesDone[{{edgeList, {i}}}] && EdgesDone[g, {i}],
        Shifts[{{g2}} + (EdgeFunctions[{{g2}} / t = lens[{{g2}}]; EdgesDone[g, {i}] = True; g2 = SetVertexWeights[g2, {edgeList[{{g2}}}, (Shifts[{{g2}}])]; VerticesDone[{{edgeList, {i}}}] = True;]
    ];
  i++
  ];
  g2 = SetVertexLabels[g2],
  Table["n = " <> ToString[GetVertexWeights[g2][i]], StandardForm <> ", A = " <> 
    ToString[PointMasses[g2][i]], StandardForm <> ", i, V[g2]]];
  g2 = SetEdgeLabels[g2],
  Table["n = " <> ToString[EdgeFunctions[g2]], StandardForm <> ", A = " <> 
    ToString[EdgeMeasures[g2][i]], StandardForm <> ", i, M[g2]]];
  g2 = SetGraphOptions[0, VertexLabel -> True, EdgeLabel -> True,
    VertexLabelPosition -> LowerRight, VertexLabelColor -> Blue,
    TextStyle -> {FontSize -> 12};
  ShowGraphArray[{g1, g2}, PlotRegion -> {{0, 1}, {0, 1}}]
7.6. Generator for General Case.

```math
\text{g = Graph}([[[1, 2]], [[2, 3]], [[2, 4]], [[3, 4]]],
[[[0, -1]], [[0, 0]], [[-1, 1]], [[1, 1]]], \text{EdgeDirection} \rightarrow \text{On});
\text{ShowGraph} (g = \text{StylizeGraph}([\text{NumberedEdges}[g]], \text{VertexNumber} \rightarrow \text{True},
\text{EdgeLabel} \rightarrow \text{True})
```

GraphQL

```math
(= \text{SetGroundNodes} \rightarrow \text{GroundNodes} = \{1\};
\text{If} [\text{Range}[V[g]]] \text{If} \text{GroundNodes == GroundNodes, "GroundNodes are OK",}
"\text{Error: GroundNodes Invalid!!!}"

\text{GroundNodes are OK}

\text{g0 = NumberedEdges[Contract[g, GroundNodes]]};
\text{ShowGraph} [g0, \text{VertexNumber} \rightarrow \text{True}, \text{EdgeLabel} \rightarrow \text{True}]`
(⇒ This will serve to hold the edge functions, ideally we only want to consider C^1 functions
the user may extend the graph as necessary, but it might still work). The size of this list must be
M[q0]; currently we only allow for NON–
NEGATIVE measures. An edge is parameterize in terms of 't' starting from the tail of the directed edge ⇒)

EdgeFunctions = FullSimplify[1, 1 - e - 1 + 1, 2 t, t];
(⇒ Here we verify normalize the EdgeFunctions and precompute a few things. ⇒)

EdgeFunctions = Table[EdgeFunctions[i, j, k, 0, {}];
EdgeDerivatives = D[EdgeFunctions, EdgeMeasures = -D[EdgeDerivatives, Edges];
Module[
  {DirectionalDerivatives =
    Flatten[
      Table[{{EdgeDerivatives[i, j, k, 0, k, 0, 1, i, j, 0]}],
      {i, M[q0]}],
      VertexMat = Table[Flatten@Table[If[Edges[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0, j, 0]}],
      , {i, M[q0]}]],
      PointMasses = -{VertexMat * DirectionalDerivatives};
    Module[
      EdgesDone = Table[False, {i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0}, EdgesDone = Table[False, {i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0}]},
      edgeList = Edges[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0],
      Shifts = Table[0, {i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0}],
      lens = GetEdgeLengths[[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]],
      q2 = SetVertexWeights[0, q2, [0, 1, j, 0, i, 0, 1, 2, i, j, 0]], EdgesDone[q2, {edgesList[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]}],
      FullSimplify[Shifts[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]]];
      VerticesDone[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0] = True;
      (⇒ If edgesDone[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0] && EdgesDone[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0],
      Shifts[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0] = SetVertexWeights[0, q2, {edgesList[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]}];
      EdgesDone[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]] = True;
      FullSimplify[Shifts[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0]]];
      VerticesDone[i, j, k, 0, 1, j, 0, i, 0, 1, 2, i, j, 0] = True;
      ];
      i++];
      EdgeFunctions = FullSimplify[EdgeFunctions + Shifts];
    ]}
References


Department of Mathematics, University of Michigan, Ann Arbor, MI 48109-1109, USA

E-mail address: dalli@umich.edu