A MAPLE APPLICATION FOR TESTING SELF-ADJOINTNESS ON QUANTUM GRAPHS

HELENE DALLMANN AND STEVEN COULTER

Abstract. In this paper we consider linear ordinary elliptic differential operators with smooth coefficients on finite quantum graphs. We discuss criteria for the operator to be self-adjoint. This involves conditions on matrices representative of the boundary conditions at each vertex. The main point is the development of a Maple application to test these conditions.

1. Introduction

The question addressed in this paper is that of deciding via a Maple application whether a system of elliptic ordinary differential operators on a quantum graph subject to given coupling conditions at the vertices is self-adjoint.

Quantum graphs are used in studying wave propagation in branching media, electron propagation in multiply connected media, and many other problems in the disciplines of mathematics, biology, chemistry, physics, and engineering. The survey article [6] gives an excellent account of quantum graph models with examples from different areas of application. Many of the problems require or assume self-adjoint operators. An application such as ours can be used to verify self-adjointness so that the conditions of the problem are met.

Furthermore, our application may serve as an educational tool to explore the notion of self-adjointness of differential operators (subject to boundary conditions). Standard methods to find solutions to partial differential equations such as separation of variables and Fourier series that are presented in undergraduate partial differential equations classes fundamentally rely on self-adjointness of the problem. Those methods work, after all, because of the Spectral Theorem, which (over the reals) holds precisely for self-adjoint operators. In fact, the vast majority of spectral theory is concerned with, and is limited to, self-adjoint operators. These operators constitute the most important class that appears in models, and it may possibly be surprising for a student to see how unstable this class actually is. Perturbations, however small, in the coefficients of the operator and the boundary conditions may turn a self-adjoint operator into a non self-adjoint one, and therefore many familiar techniques to attack problems (such as separation of variables) become unavailable.

Quantum graphs are particularly suitable objects for students to explore the interplay between the analysis of geometric differential operators (like the Laplace or

2010 *Mathematics Subject Classification.* 34B45.

Key words and phrases. Selfadjoint Differential Operators, Quantum Graphs, Symplectic Vector Spaces.

Penn State – Göttingen Summer School 2010 supported by the National Science Foundation, Grant DMS-0963728

Faculty advisors: Thomas Krainer and Michael Weiner, Penn State Altoona, 3000 Ivyside Park, Altoona, PA 16601, U.S.A. Email: krainer@psu.edu and mdw8@psu.edu.

the Dirac operator) and the topology of the underlying space. Quantum graphs in general can have non-trivial topology, and yet they are 1-dimensional and therefore methods from undergraduate linear algebra and differential equations are applicable. For example, to illustrate that interplay in the context of our work, consider a graph with a single edge and on it the Dirac operator $D_x = -i\frac{d}{dx}$. If the graph has two vertices connected by that edge, then it is impossible to impose (local) boundary conditions at each vertex to make D_x self-adjoint. However, if the graph is a loop (so has only one vertex), then D_x is self-adjoint with periodic (or transmission) boundary conditions and thus represents the Dirac operator on a circle – its spectral family is historically the first and still the principal example of a Fourier basis. Note that the circle is topologically non-trivial, while the interval (2-vertex graph) is homotopic to a point. This example also shows that it may be impossible to associate self-adjoint (local) vertex conditions for an operator on a quantum graph. Our Maple application not only tests given vertex conditions, but is also capable of analyzing whether a system of differential operators on a quantum graph allows self-adjoint vertex conditions at all.

While the notion of self-adjointness is conceptually not hard to grasp, it can be a considerable computational challenge to actually verify it. For example, consider a quantum graph with vertices where many edges meet, and impose vertex conditions that have a high degree of coupling. Verification of self-adjointness in such a situation is computationally extremely involved. A systematic approach aided by computer algebra software is desirable, and our Maple application provides just that.

The paper is organized in three parts. In the first part we recall the most important definitions and specify the setting of the problem.

The second part deals with an algebraic approach to analyze self-adjointness that is based on symplectic linear algebra [2, 3]. We discuss results that allow us to determine suitable algebraic conditions to analyze the problem. Furthermore, some simple examples are presented.

The first two parts provide the necessary foundation for the Maple application that we propose in the third part of the paper. This section includes a thorough explanation and discussion of the implementation. The code itself is available on the internet at

http://math.aa.psu.edu/∼summerschool

Section 3.2 provides detailed information about accessing the files and the usage of the Maple application. The code can also be found in the appendix.

1.1. Quantum Graphs.

A graph Γ is defined as a set of vertices $\mathcal{V} = \{\mathcal{V}_p\}$ along with a set of edges $\mathcal{E} = \{e_i\}$, with each edge e_i connecting a pair of vertices. We will consider the case only when the sets $\mathcal{V}$ and $\mathcal{E}$ contain finitely many elements. Let $|\mathcal{V}|$ and $|\mathcal{E}|$ denote the total number of vertices and edges, respectively.

When we consider each edge as an interval, we can view a graph as a collection of disjoint intervals where the vertices are formed by the gluing together of endpoints of different intervals. We will consider without loss of generality the case when each interval (edge) e_i of the graph Γ is of unit length, $e_i = [0, 1]$ for $i = 1, 2, \ldots$. The space of measurable and square integrable functions along an edge e_i is denoted by

$$L^2(e_i) = \left\{ f \text{ defined on } e_i \;\middle|\; \int_{e_i} |f(x)|^2\,dx < \infty \right\}.$$

The sum of the L^2 spaces of each edge creates the L^2 space for the entire graph,

$$L^2(\Gamma) = \bigoplus_{i=1}^{|\mathcal{E}|} L^2(e_i).$$

The application of a differential operator A_i to each edge e_i turns the graph into a *quantum graph.*

Definition 1.1. A *quantum graph* is a graph where each edge e_i is associated with a copy of the unit interval $[0,1]$ and has defined upon it a differential operator A_i.

The survey [6] contains a detailed description of quantum graphs.

1.2. **Maximal and Minimal Domains.** Consider a linear elliptic ordinary differential operator with smooth, possibly complex-valued coefficients

$$A = \sum_{j=0}^{n} a_j(x) \frac{d^j}{dx^j}, \tag{1.2}$$

where $a_j(x) \in C^\infty([0,1])$ and $a_n(x) \neq 0$ everywhere on $[0,1]$. We assume that A is formally self-adjoint or symmetric, i.e. $A = A^\#$ where $A^\#$ is the formal adjoint of A and denotes the differential operator given by

$$\langle Af, g\rangle = \langle f, A^\# g\rangle \text{ for } f, g \in C_c^\infty(0,1)$$

under the complex inner product

$$\langle f, g\rangle = \int_0^1 f\bar{g}\, dx, \quad f,\, g \in L^2([0,1])\,. \tag{1.3}$$

Here $C_c^\infty(0,1)$ denotes the set of all C^∞-smooth functions that have compact support contained in $(0,1)$. The operator $A = D_x^n$ for every $n = 1,2,3,\ldots$ is an example that satisfies these conditions, where $D_x = -i\frac{d}{dx}$.

We furthermore define $\mathcal{D}_{\max}$ and $\mathcal{D}_{\min}$ to be the maximal and minimal L^2 domains of A as follows:

$$\mathcal{D}_{\max}(A) := \left\{u \in L^2([0,1]) \,|\, \exists v \in L^2([0,1])\colon \langle v, \varphi\rangle = \langle u, A^\#\varphi\rangle \;\forall \varphi \in C_c^\infty(0,1)\right\}. \tag{1.4}$$

For $u \in \mathcal{D}_{\max}(A)$ define $A_{\max}u = v$, where $v \in L^2([0,1])$ satisfies the condition in (1.4). This definition gives rise to the *maximal extension* $A_{\max} : \mathcal{D}_{\max}(A) \to L^2([0,1])$ of A. Note that $C_c^\infty(0,1)$ is dense in $L^2([0,1])$, which implies that $A_{\max}$ is well-defined. It is custom to drop the subscript from $A_{\max}$ and simply write A. We will use both notations in the sequel and leave the subscript where we believe that greater clarity is warranted.

The minimal L^2-domain of A can then be defined as

$$\mathcal{D}_{\min}(A) := \left\{u \in \mathcal{D}_{\max}(A) \,|\, \langle Au, v\rangle = \langle u, A^\# v\rangle \;\forall v \in \mathcal{D}_{\max}(A^\#)\right\}. \tag{1.5}$$

Let $A_{\min} = A_{\max}\big|_{\mathcal{D}_{\min}(A)}$ denote the restriction of $A_{\max}$ to $\mathcal{D}_{\min}(A)$.
In addition we call the operator $A_{\mathcal{D}} = A\,|_{\mathcal{D}} : \mathcal{D}_{\max}(A) \supseteq \mathcal{D} \to L^2$ *closed,* if and only if the graph $\Gamma_{A_{\mathcal{D}}} = \{(u, Au) \,|\, u \in \mathcal{D}\} \subset L^2 \times L^2$ is closed.

We remark here that $\mathcal{D}_{\max}(A)$ determines the elements $u \in L^2([0,1])$ such that $Au \in L^2([0,1])$, while $\mathcal{D}_{\min}(A)$ is the smallest subspace of the maximal domain that contains $C_c^\infty(0,1)$ such that $A_{\min}\colon \mathcal{D}_{\min}(A) \to L^2([0,1])$ is closed.

Under our present assumptions on the operator A it can be shown that the maximal domain $\mathcal{D}_{\max}(A)$ is exactly the Sobolev space $H^n[0,1]$ and $\mathcal{D}_{\min}(A) =$

$H_0^n([0,1])$, the H^n-distributions with compact support in $[0,1]$. Consequently, if the operator A is as considered in this paper, the minimal and maximal L^2-domains are independent of A and only see its order. Moreover, the extension $A_{\mathcal{D}}$ of A is closed for all intermediate domains $\mathcal{D}_{\min}(A) \subset \mathcal{D} \subset \mathcal{D}_{\max}(A)$ between the minimal and the maximal L^2-domains of A.

A comment on nomenclature is in order: It is custom to think of the operator A initially as only acting on $C_c^\infty(0,1)$. Extensions of A are then linear operators that act on larger spaces (typically called domains), but restrict to the given action of A on $C_c^\infty(0,1)$. The maximal extension $A_{\max}$ is such an extension with domain $\mathcal{D}_{\max}(A)$ that is obtained using the L^2-inner product, and so is $A_{\min}$ with domain $\mathcal{D}_{\min}(A)$. The extensions of A that are considered here all have intermediate domains $\mathcal{D}_{\min}(A) \subset \mathcal{D} \subset \mathcal{D}_{\max}(A)$. By Proposition 1.7 further below, these are precisely the ones that are obtained from imposing boundary conditions. These extensions are frequently referred to also as realizations of the operator A (subject to boundary conditions).

We consider the differential operator A on a closed interval $I = [a,b] \subset \mathbb{R}$. Without restriction we assume $[a,b] = [0,1]$. We define the *left endpoint space* to be

$$E_l(A) = \{\omega_l u + \mathcal{D}_{\min} \mid u \in \mathcal{D}_{\max}\},$$

where $\omega_l \in C^\infty(I)$ denotes a left cut-off function such that $\omega_l \equiv 1$ near the left endpoint and $\omega_l \equiv 0$ near the right endpoint. The right endpoint space $E_r(A)$ is defined analogously with a cut-off function ω_r that vanishes at the left endpoint. Then the following holds:

$$\mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A) = E_l(A) \oplus E_r(A) \tag{1.6}$$

We thus have the following proposition:

Proposition 1.7. *The map*

$$T_l : C^\infty([0,1]) \ni f \longmapsto \big(f(0), f'(0), f''(0), \ldots, f^{(n-1)}(0)\big) \in \mathbb{C}^n$$

extends to a continuous linear map $T_l : \mathcal{D}_{\max}(A) \longrightarrow \mathbb{C}^n$ that vanishes on $\mathcal{D}_{\min}(A)$, hence factors to a map

$$\hat{T}_l : \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A) \to \mathbb{C}^n.$$

The map $\hat{T}_l$ is onto, and its kernel is $E_r(A)$. Consequently, $\hat{T}_l$ induces an isomorphism $E_l(A) \cong \mathbb{C}^n$. An analogous statement holds for the right endpoint space.

The inverse map is given by

$$\hat{T}_l^{-1} : \mathbb{C}^n \to E_l(A),$$
$$\hat{T}_l^{-1}(a_0, \ldots, a_{n-1}) = \omega(x) \sum_{j=0}^{n-1} \frac{a_j}{j!} x^j + \mathcal{D}_{\min}(A),$$

where ω is a C^∞-function on $[0,1]$ such that

$$\omega(0) = 1, \quad \frac{d^k\omega}{dx^k}(0) = 0,\ k = 1, \ldots, n-1,$$
$$\frac{d^j\omega}{dx^j}(1) = 0,\ j = 0, \ldots, n-1.$$

We refer to the monographs [1, 2, 7, 8] for background on the above.

1.3. **Symplectic Form.**

The primary focus will be on self-adjointness of extensions of operators. To this end, we will utilize the symplectic formalism as discussed, e.g., in [2]. The precise context is elaborated in greater detail in Section 2 below. At this point, we recall some of the basics of the symplectic formalism, especially in the context of quantum graphs.

A *complex symplectic vector space* is a complex vector space V equipped with a *complex symplectic form*

$$[\cdot,\cdot] : V \times V \to \mathbb{C}, \tag{1.8}$$

that satisfies for all $u, v, w \in V$ and $\lambda \in \mathbb{C}$ the following conditions:

(1) $[u + v, w] = [u, w] + [v, w]$.
(2) $[\lambda u, v] = \lambda[u, v]$.
(3) $[u, v] = -\overline{[v, u]}$.

That is to say, the symplectic form is conjugate bilinear and skew-Hermitian. We call the form *non-degenerate* if it satisfies

(4) $[u, v] = 0$ for all $v \in V \;\Leftrightarrow\; u = 0$.

Under our standing assumptions on the formally self-adjoint operator A on the interval $[0, 1]$ as stated at the beginning of Section 1.2, we obtain that the Hermitian inner product (1.3) gives rise to the skew-Hermitian form

$$\mathcal{D}_{\max} \times \mathcal{D}_{\max} \ni (u, v) \mapsto \int_0^1 Au \cdot \overline{v}\, dx - \int_0^1 u \cdot \overline{Av}\, dx = \langle Au, v\rangle - \langle u, Av\rangle, \tag{1.9}$$

which translates into a nondegenerate complex symplectic form

$$[\cdot,\cdot]_A \colon V \times V \to \mathbb{C} \tag{1.10}$$

where in this case V denotes the quotient space

$$V = \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A).$$

The direct sum in (1.6) is in fact a symplectic orthogonal direct sum. This means that, in addition to the sum being direct, we have $[u_l, u_r]_A = 0$ for all $u_l \in E_l(A)$ and $u_r \in E_r(A)$.

1.4. **Symplectic Spaces on Quantum Graphs.**

Now, consider a finite quantum graph Γ with the operator $A = \{A_i\}$, where each A_i is defined on the edge e_i, is formally self-adjoint, and elliptic. The quotient space $\mathcal{D}_{\max}(A_i)/\mathcal{D}_{\min}(A_i)$ associated with every operator A_i is a symplectic vector space when equipped with the form (1.10) that is induced by (1.9).

The maximal and minimal domains of A on the graph are by definition the direct sum of the maximal and minimal domains of each A_i.

$$\mathcal{D}_{\max}(A) = \bigoplus_{j=1}^{|\mathcal{E}|} \mathcal{D}_{\max}(A_j), \qquad \mathcal{D}_{\min}(A) = \bigoplus_{j=1}^{|\mathcal{E}|} \mathcal{D}_{\min}(A_j).$$

Let $V = \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A)$, analogously to the single interval case discussed above. V carries a symplectic form which is induced by the symplectic forms on the quotient spaces of each edge. This symplectic form will also be denoted by $[\cdot,\cdot]_A$. Thus, V is the symplectic orthogonal direct sum of the left and right endpoint spaces of the operator A_i on each edge e_i:

$$V = \bigoplus_{i=1}^{|\mathcal{E}|} \mathcal{D}_{\max}(A_i)/\mathcal{D}_{\min}(A_i) = \bigoplus_{i=1}^{|\mathcal{E}|} (E_l(A_i) \oplus E_r(A_i)) \,. \tag{1.11}$$

1.5. **Vertex conditions.** Let $A = \{A_i\}$ be an operator on the graph Γ as above, and consider a linear map

$$B : \mathcal{D}_{\max}(A) \to \mathbb{C}^k$$

for some $k \in \mathbb{N}$ that vanishes on $\mathcal{D}_{\min}(A)$. In particular, B induces a map $\hat{B} : V \to \mathbb{C}^k$, where $V = \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A)$. With such a map B we can associate an abstract boundary value problem (BVP) as follows:

$$\text{Find } u \in \mathcal{D}_{\max}(A)\colon \begin{cases} Au = f \in L^2(\Gamma) \\ Bu = 0 \in \mathbb{C}^k \,. \end{cases} \tag{1.12}$$

Then there is an obvious relation between (BVP) and some extension $A_{\mathcal{D}_B}$ for some domain $\mathcal{D}_{\min} \subset \mathcal{D}_B \subset \mathcal{D}_{\max}$ given by

$$\mathcal{D}_B = \{u \in \mathcal{D}_{\max}(A) \,|\, Bu = 0\} \,,$$

and (1.12) is equivalent to the problem

$$\text{find } u \in \mathcal{D}_B\colon Au = f \,.$$

In view of our ellipticity assumptions on the operators A_i and Proposition 1.7, we conclude that B (or $\hat{B}$) expresses in fact linear relations between the values of functions (and their derivatives) that are defined on Γ at the vertices. Thus, every abstract boundary value problem is in fact a boundary value problem in the classical sense. In this general setup, values of functions at different vertices could be subjected to linear relations by $\hat{B}$. If this is not the case, B (or $\hat{B}$) is said to determine *local vertex conditions* as defined below.

Definition 1.13. For every vertex p of the graph Γ define the *vertex space*

$$\mathcal{V}_p = \bigoplus (\text{endpoint spaces connected at vertex } p\ ) \,.$$

A domain $\mathcal{D}_{\min} \subset \mathcal{D} \subset \mathcal{D}_{\max}$ for A on Γ is said to be specified by *local vertex conditions* if

$$\mathcal{D}/\mathcal{D}_{\min} = \bigoplus_{p \text{ vertex}} U_p$$

for subspaces $U_p \subset \mathcal{V}_p$.

The symplectic vector space $\mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A)$ on Γ can be represented as the symplectic orthogonal direct sum of the vertex spaces:

$$V = \bigoplus_{p \text{ vertex}} \mathcal{V}_p \,. \tag{1.14}$$

Note that in case of local vertex conditions, we impose linear relations on the collection of boundary values of the respective functions on Γ involving only the endpoints of edges that meet at every single vertex. We will restrict our discussion to these local conditions since the non-local case can be reduced to the local one [6].

2. Algebraic Interpretation of Self-Adjointness

We begin this section by defining the adjoint of an operator $A = \{A_i\}$ on Γ. Our standing assumption is that each A_i is elliptic and formally self-adjoint.

Definition 2.1. Consider the operator $A_{\mathcal{D}}$ with domain $\mathcal{D}_{\min} \subset \mathcal{D} \subset \mathcal{D}_{\max}$. Then the *adjoint* of $A_{\mathcal{D}}$ is defined by $(A_{\mathcal{D}})^* = A^{\#}_{\mathcal{D}^*}$, the formal adjoint operator $A^{\#}$ with domain $\mathcal{D}^* = \{v \in \mathcal{D}_{\max} \,|\, [u, v]_A = 0 \text{ for all } u \in \mathcal{D}\}$.
In particular, if A is formally self-adjoint as we assume here, i.e. $A = A^{\#}$, $A^*_{\mathcal{D}} = A_{\mathcal{D}^*}$ holds. The operator $A_{\mathcal{D}}$ is called *self-adjoint*, if $\mathcal{D} = \mathcal{D}^*$.

Since we assume A to be formally self-adjoint, we obtain the following from the definitions of $\mathcal{D}_{\min}$ and $\mathcal{D}_{\max}$:

$$(A_{\min})^* = A_{\max} \ , \ (A_{\max})^* = A_{\min} \, .$$

In this paper, we will deal with the following issues:

(1) Does A admit a self-adjoint extension at all?
(2) Does A admit self-adjoint extensions subject to local vertex conditions on Γ?
(3) How can one effectively recognize self-adjoint extensions?

2.1. Conditions for Self-Adjointness.

In this section we will discover that the existence of self-adjoint extensions $A_{\mathcal{D}}$ requires the selection of a domain $\mathcal{D}$ where $\mathcal{D}_{\min} \subset \mathcal{D} \subset \mathcal{D}_{\max}$, and $U = \mathcal{D}/\mathcal{D}_{\min}$ is a maximally isotropic subspace of the symplectic vector space $V = \mathcal{D}_{\max}/\mathcal{D}_{\min}$. However, self-adjointness is actually even stronger than that. Furthermore, we will find a condition which guarantees that a maximally isotropic subspace U yields a self-adjoint extension. The arguments are largely based on [2].

Definition 2.2. For a subspace $U \subseteq V$ we define the *symplectic orthogonal* U^* of U by

$$U^* = \{v \in V | [u, v]_A = 0 \ \forall u \in U\}.$$

Moreover, U is called *isotropic* if

$$U \subseteq U^*.$$

Let $\tilde{U} \subseteq V$ be isotropic. If $U \subseteq \tilde{U} \ \Rightarrow \ U = \tilde{U}$, then U is called *maximally isotropic.*

Theorem 2.3. *Let $U \subseteq V$ and U^* its symplectic orthogonal. Then the following holds:*

$$\dim U + \dim U^* = \dim V \, . \tag{2.4}$$

We now observe a connection between the adjoint operator and isotropic subspaces:

Proposition 2.5. *Let $U = \mathcal{D}/\mathcal{D}_{\min}$. If A acts with domain $\mathcal{D}$, then the domain $\mathcal{D}^*$ of the adjoint of $A_{\mathcal{D}}$ satisfies $\mathcal{D}^*/\mathcal{D}_{\min} = U^*$. Therefore self-adjointness of $A_{\mathcal{D}}$ means $U = U^*$, which is equivalent to $\mathcal{D} = \mathcal{D}^*$.*

We remark here that if A allows a self-adjoint extension, it is a necessary condition for V to be even-dimensional.

We will now discuss isotropic subspaces of a *general complex symplectic vector space* $(W, [\cdot, \cdot])$ equipped with a non-degenerate symplectic form $[\cdot, \cdot]$.

Definition 2.6. Let $(W, [\cdot,\cdot])$ be a symplectic vector space. The *symplectic invariants* of $(W, [\cdot,\cdot])$ are the numbers $p(W)$ and $q(W)$ given by

$$p(W) := \max\{\dim U \mid U \subseteq W, \text{ and } \Im[u,u] \geq 0 \text{ for all } u \in U\}$$
$$q(W) := \max\{\dim U \mid U \subseteq W, \text{ and } \Im[u,u] \leq 0 \text{ for all } u \in U\}.$$

Note that whenever U is isotropic we have that

$$\dim(U) \leq \min\{p(W), q(W)\}.$$

Equality holds for maximally isotropic subspaces (see e.g. [2]).

We can discover the symplectic invariants in the following manner. Let $n = \dim(W)$ and select a basis $\{v_1, \ldots, v_n\}$ of W. Then we can express any $u, w \in W$ as

$$u = \sum_{j=1}^{n} a_j v_j, \qquad w = \sum_{j=1}^{n} b_j v_j.$$

Writing these into the symplectic form we have that

$$[u,w] = \left[\sum_{j=1}^{n} a_j v_j, \sum_{j=1}^{n} b_j v_j\right] = \begin{pmatrix} a_1 & \ldots & a_n \end{pmatrix} H \begin{pmatrix} \bar{b}_1 \\ \vdots \\ \bar{b}_n \end{pmatrix},$$

where $H = ([v_i, v_j]_A)_{1\leq i,j\leq n}$. Property (3) of the symplectic form $[\cdot,\cdot]$ in (1.8) gives us that H is a skew-Hermitian matrix, which implies H is normal and thus diagonalizable. Since the symplectic form is assumed to be non-degenerate, we can find an invertible matrix Q such that

$$QHQ^* = \begin{pmatrix} \mathrm{i}I_{\tilde{p}} & 0 \\ 0 & -\mathrm{i}I_{\tilde{q}} \end{pmatrix}. \tag{2.7}$$

Lemma 2.8. *$\tilde{p} = p(W)$ and $\tilde{q} = q(W)$.*

Proof. Let $\{w_1, \ldots, w_n\}$ be the basis of W as determined by the transformation matrix Q such that $([w_i, w_j]_A)_{1\leq i,j\leq n}$ is the matrix (2.7). Let $U = \operatorname{span}\{w_1, \ldots, w_{\tilde{p}}\}$. Then we have that

$$\forall u \in U\backslash\{0\}, \Im[u,u] \geq 0 \tag{2.9}$$

and so $p(W) \geq \tilde{p}$. Now take $U' = \operatorname{span}\{w_{\tilde{p}+1}, \ldots, w_n\}$. Obviously $\dim U' = \tilde{q}$ and for any subspace $\tilde{W}$ of W where the condition (2.9) is met for $w \in \tilde{W}$ we have that $\tilde{W} \cap U' = \{0\}$. Thus

$$\dim \tilde{W} \leq n - \tilde{q} = \tilde{p}.$$

Since U is one such $\tilde{W}$ we have exactly that $p(W) = \tilde{p}$, and we have similar proof for the statement that $\tilde{q} = q(W)$. □

This allows us to state the following corollary which we will make use of later on.

Corollary 2.10. *If $W = W_1 \oplus W_2$ is a symplectic orthogonal direct sum, then $p(W) = p(W_1) + p(W_2)$ and $q(W) = q(W_1) + q(W_2)$.*

We are thus prepared to state a theorem (see [2]) that provides a set of conditions that can be checked to allow for a self-adjoint extension of an operator.

Theorem 2.11. *A symplectic vector space W allows for subspaces U with $U = U^*$ if and only if the symplectic invariants are equal.*

This implies that the operator $A = \{A_i\}$ on the quantum graph has a self-adjoint extension if and only if the symplectic invariants of the quotient space $V = \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A)$ are equal. The subspaces that satisfy $U = U^*$ are precisely the maximally isotropic subspaces of V in this case. So V is necessarily even-dimensional and

$$\dim U = \frac{\dim V}{2}\,. \tag{2.12}$$

Due to Corollary 2.10 we can make another point here for quantum graphs. If each single-interval operator A_i has equal symplectic invariants, the operator A on a graph admits self-adjoint extensions, but those extensions are not necessarily ones that are subject to local vertex conditions. Local vertex conditions require the domain $\mathcal{D}/\mathcal{D}_{\min}$ to be a direct sum of subspaces U_p (where p denotes a vertex) and each U_p has to satisfy $U_p = U_p^*$. This gives rise to the following result.

Theorem 2.13. *Self-adjoint extensions subject to local vertex conditions exist if and only if the symplectic invariants of each vertex space are equal. The symplectic invariants of each vertex space are the sums of the symplectic invariants of all the endpoint spaces forming the vertex space.*

In the following we will provide verifiable criteria to check whether a subspace U of V is maximally isotropic.

We now consider V to be of even dimension $2n$ and $\dim U = n$ as well as $p(V) = q(V)$. This assumption on the symplectic invariants is necessarily required to hold for the quotient space V.

Let $\{v_1, \dots, v_{2n}\}$ be a basis of V and $\{u_1, \dots, u_n\}$ a basis of U. Since $U \subseteq V$, the basis vectors can be written as

$$u_i \;=\; \sum_{j=1}^{2n} a_{ij} v_j, \qquad i = 1, \dots, n, \quad a_{ij} \in \mathbb{C}$$

It is enough test the condition of isotropy (Definition 2.2) for each of the basis elements. We have for $1 \le i, k \le n$

$$0{=}[u_i, u_k]_A = \left[\sum_{j=1}^{2n} a_{ij} v_j, \sum_{l=1}^{2n} a_{kl} v_l\right]_A = \begin{pmatrix} a_{i,1} & \dots & a_{i,2n}\end{pmatrix} H \begin{pmatrix} \bar a_{k,1} \\ \vdots \\ \bar a_{k,2n}\end{pmatrix}\,,$$

where $H = ([v_i, v_j]_A)_{1 \le i,j \le 2n}$.

We can perform a basis transformation as explained earlier in this paper such that H is without loss of generality of the form

$$H = \begin{pmatrix} \mathrm{i}I_n & 0 \\ 0 & -\mathrm{i}I_n \end{pmatrix}\,. \tag{2.14}$$

In this case we calculate

$$\begin{aligned} 0 \;&=\; [u_i, u_k] \;=\; (a_{i,1} \;\dots\; a_{i,2n}) \begin{pmatrix} \mathrm{i}I_n & 0 \\ 0 & -\mathrm{i}I_n \end{pmatrix} \begin{pmatrix} \bar a_{k,1} \\ \cdot \\ \cdot \\ \cdot \\ \bar a_{k,2n} \end{pmatrix} \\ &=\; \mathrm{i} \cdot (a_{i,1}\bar a_{k,1} + \dots + a_{i,n}\bar a_{k,n} - a_{i,n+1}\bar a_{k,n+1} - \dots - a_{i,2n}\bar a_{k,2n})\,. \end{aligned}$$

If we write $(a_{ij})_{1\leq i\leq n,1\leq j\leq 2n} = \begin{pmatrix} C & D\end{pmatrix}$ with two $n\times n$ matrices C and D, this gives us exactly the relation

$$\begin{aligned} \left(C\bar{C}^T\right)_{i,k} &= \left(D\bar{D}^T\right)_{i,k} \text{ for all } i,k \\ \Leftrightarrow \quad CC^* &= DD^*\,. \end{aligned} \tag{2.15}$$

So U is isotropic under this condition and indeed maximally isotropic if the dimension of U is the maximal possible one, which is exactly the case if $\begin{pmatrix} C & D\end{pmatrix}$ is of full rank (n).

In order to be able to compare with other results, we will need the representation of H with respect to a so called *standard canonical symplectic basis*, i.e., $\{v_1,\ldots,v_{2n}\}$ is such that H is of the form

$$H = \begin{pmatrix} 0 & I_n \\ -I_n & 0\end{pmatrix}\,.$$

A symplectic vector space has such a basis if and only if the symplectic invariants are equal ([2]). Thus, we are left with the following relationship between the coefficients a_{ij}:

$$a_{i,1}\bar{a}_{k,n+1} + \cdots + a_{i,n}\bar{a}_{k,2n} = a_{i,n+1}\bar{a}_{k,1} + \cdots + a_{i,2n}\bar{a}_{k,n}\,. \tag{2.16}$$

Following the same method as above, we obtain in this case that

$$CD^* = DC^*\,. \tag{2.17}$$

2.2. **Application to Quantum Graphs.** We can now utilize the result of Section 2.1 and apply it to a finite quantum graph in order to determine whether the given set of differential operators and boundary conditions give rise to a self-adjoint extension. Since V is representable as the symplectic orthogonal direct sum of the vertex spaces, a domain $\mathcal{D}$ as determined by local vertex conditions gives rise to a subspace $U\subseteq \mathcal{D}_{\max}(A)/\mathcal{D}_{\min}(A)$ that is a symplectic orthogonal direct sum of subspaces $U_p\subset \mathcal{V}_p$ of the vertex spaces $\mathcal{V}_p$.

$$U = \bigoplus_{p\text{ vertex}} U_p.$$

Self-adjointness means that $U_p^* = U_p$ for all p. This entails that the conditions for self-adjointness can be checked in terms of matrices representative of the local vertex conditions of each vertex alone, rather than the matrix representative of the conditions on the entire graph.

To begin, let us see how the subspace U_p is determined by the local vertex conditions. Let the operators A_i be of order n_i, thus each endpoint space is n_i-dimensional and the vertex conditions concerning the endpoints of e_i can only consist of the function values up to the $(n_i-1)^{\text{st}}$ derivative, see Proposition 1.7.

If d endpoints are connected at a vertex p, let $n_p = \sum\{n_i | f_i \text{ meets at } p\}$ represent the total dimension of this vertex space $\mathcal{V}_p$. In order to discuss self-adjointness subject to local vertex conditions, each n_p must necessarily be even. The local homogeneous boundary conditions can then be represented by the linear system $T_pf_p = 0$, where T_p is a $n_p/2\times n_p$-matrix of maximal rank $n_p/2$ and f_p is a vector whose entries consist of the boundary values of each function f_i at the endpoint $k_i\in\{0,1\}$ of the edge e_i meeting at the vertex p. Enumerating the functions $1,\ldots,d$ (thereby allowing each function to be represented at most twice since we allow both interval endpoints to be connected to the same vertex), we have the representation

$$f_p = \begin{pmatrix} f_1^{(0)}(k_1) & \dots & f_1^{(n_1-1)}(k_1) & \dots & f_d^{(0)}(k_d) & \dots & f_d^{(n_d-1)}(k_d) \end{pmatrix}^T .$$

Thus, the subspace $U_p \subset \mathcal{V}_p$ of dimension $n_p/2$ is defined by the system

$$T_p f_p = 0.$$

We let T denote the matrix representative of the direct sum of each of the matrices T_p. Relation (2.15) requires $\mathcal{V}_p$ to be even dimensional, so as already said we require that n_p be even, and hence $n_p/2$ will be integer valued.

With this theoretical background we can now collect all conditions needed for self-adjoint extensions and formulate this theorem:

Theorem 2.18. *Let A be an elliptic formally self-adjoint operator on a quantum graph. A choice of domain $\mathcal{D}_{\min} \subset \mathcal{D} \subset \mathcal{D}_{\max}$ for A gives rise to a self-adjoint extension for A subject to local vertex conditions if and only if $U = \mathcal{D}/\mathcal{D}_{min}$ is a direct sum of spaces $U_p \subset \mathcal{V}_p$ that satisfy $U_p^* = U_p$ for each vertex p , where the symplectic orthogonal U_p^* is taken within the symplectic vector space $\mathcal{V}_p$ for each p.*

The key to utilizing matrix algebra to verify the conditions stated in this theorem is a basis transformation to diagonalize the matrix representing the symplectic form on $\mathcal{V}_p$, see (2.7) and (2.14), and then checking (2.15) for the matrix representing the vertex condition in the new basis. See Section 3 (explanation of the algorithm) for further details.

2.3. Common Vertex Conditions.

Whenever A is the one dimensional Laplacian,

$$A = -\Delta = -\frac{d^2}{dx^2},$$

acting on every edge, there are three common types of local vertex conditions that result in self-adjoint realizations of an operator A on a graph Γ. As described in [6], they are as follows.

2.3.1. *Kirchhoff Conditions.*

The Kirchhoff Conditions require that f is continuous on the entire graph, meaning that at each vertex the function values are equal, and at each vertex the sum of the *outgoing* derivative values (i.e. taken in a direction away from the vertex) vanish. That is to say,

$$\begin{cases} f \text{ is continuous on } \Gamma \\ \forall\, p, \quad \displaystyle\sum_{i \in E_p} \frac{df}{dx_e}(p) = 0 \end{cases}$$

2.3.2. *δ Conditions.*

The δ are similar to the Kirchoff conditions in the respect f must be continuous, but we allow the sum of derivative values to equal a multiple of the function value at the vertex.

$$\begin{cases} f \text{ is continuous on } \Gamma \\ \forall\, p, \quad \displaystyle\sum_{i \in E_p} \frac{df}{dx_e}(p) = \alpha_p f(v) \end{cases}$$

2.3.3. δ' *Conditions.* The δ' conditions reverse the roles of the function values and derivatives of the δ conditions.

$$\begin{cases} \forall\, p, \quad \frac{df}{dx_e}(p) \text{ are independent of } e \\ \\ \forall\, p, \quad \displaystyle\sum_{i \in E_p} f(p) = \alpha_p \frac{df}{dx_e}(p) \end{cases}$$

3. Maple Implementation

A Maple application was written to determine if an operator $A = \{A_i\}$ on a quantum graph subject to vertex conditions is self-adjoint, or to determine if it allows self-adjoint vertex conditions at all. The application was developed in Maple 14 and is available online at

http://math.aa.psu.edu/∼summerschool

3.1. **Algorithm.** By Theorem 2.18, in order to check for self-adjointness of an operator A on a graph, we must verify the following conditions:

(1) Each A_i is elliptic and formally self-adjoint.
(2) $\dim U = n_p/2$ for all vertices p.
(3) $CC^* = DD^*$, where $\tilde{T} = \begin{pmatrix} C & D \end{pmatrix}$ and $\tilde{T}$ is built out of T by a change of basis, see (2.15).

The algorithm therefore begins by assembling the coefficients of the differential operators upon each edge into a matrix C. The entry c_{ij} corresponds to the coefficient of $\frac{d^j}{dx^j}$ for the operator acting on edge i. First, two validation procedures are performed on each operator A_i. Let n be the order of the operator A_i. To ensure ellipticity we verify that

$$\min_{x \in [0,1]} |c_{in}(x)| > 0.$$

To test that the operator is formally self-adjoint, we test the following equality

$$A_i u(x) - A_i^{\#} u(x) = \sum_{j=0}^{n} c_{ij}(x) \frac{d^j}{dx^j} u(x) - \sum_{j=0}^{n} (-1)^j \frac{d^j}{dx^j} \left[\overline{c_{ij}}(x) u(x) \right] = 0.$$

These are the only global procedures in the application. All other operations are performed separately for each vertex.

Each vertex is processed in the order in which it is input.

For condition (3) the matrix representation $\tilde{H}$ of the symplectic form is generated for a vertex p by plugging in a basis of each endpoint space into the symplectic form. We use the Taylor basis $1, x, x^2/2, \ldots$ (or $1, (x-1), (x-1)^2/2, \ldots$, respectively) multiplied by (left or right) cut-off functions here. More specifically $\tilde{H}$ is assembled from diagonal blocks, where each block corresponds to the symplectic form computed for each operator A_i at the endpoint joined to p.

Then we compute the eigenvalues und eigenvectors of $\tilde{H}$ via Maple; so the matrix Q_1 of all eigenvectors diagonalizes $\tilde{H}$. To ensure that the resulting matrix is of the form (2.14) the matrix Q_2 normalizes the coefficients of the diagonal entries. Thus, we obtain

$$Q \tilde{H} Q^* = H$$

where $Q := Q_2 \cdot Q_1$ is an invertible matrix and H is the matrix (2.14). The algorithm checks at this point if the symplectic invariants are equal at each vertex.

The matrix T_p, as discussed in Section 2.2, is computed from the boundary conditions input by the user. Each row of T_p corresponds to a given condition. For each endpoint k of edge i meeting p, a block of columns is generated by the coefficients of $f_i(k), f_i'(k), \ldots, f_i^{(n_i-1)}(k)$. The blocks are then assembled to form the matrix. This is best illustrated through an example. Suppose three endpoints, both endpoints 0 and 1 from one interval and the endpoint 0 from a second interval, form a vertex p with the given condition

$$f_1(0) - f_1(1) = 0, \quad f_1(1) - f_2(0) = 0, \quad f_1'(0) - f_1'(1) + f_2'(0) = 0.$$

The 3×6 matrix T_p has the form

$$\begin{array}{r} \\ \text{condition 1} \\ \text{condition 2} \\ \text{condition 3} \end{array} \begin{array}{c} \begin{array}{cccccc} f_1(0) & f_1'(0) & f_1(1) & f_1(0) & f_2(0) & f_2'(0) \end{array} \\ \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

We verify that T_p has full rank, and determine the matrix N whose columns are the basis of $\ker(T)$.

At this point, we are prepared to compute the matrix $\tilde{T}$, which is the transformation of the vertex conditions into the basis appropriate to test Theorem 2.18, especially condition (3) as stated above, by taking

$$\tilde{T} = N^t Q^{-1} =: \begin{pmatrix} C & D \end{pmatrix} \tag{3.1}$$

where C, D are $n \times n$-matrices. Self-adjointness is checked by verifying that

$$CC^* - DD^* = 0.$$

3.2. **Usage.** Currently there are three files available for download:

`QG.mla`	Maple library file, the application module
`graph-sa.mw`	Maple workbook file, application source code
`graph-examples.mw`	Maple workbook file, several examples of quantum graphs

The direct link to the file `QG.mla` is

http://math.aa.psu.edu/∼summerschool/projects/QG.mla

and likewise for the other files with `QG.mla` at the end replaced by the corresponding file name.

At the very least, the `QG.mla` file is needed to run the application. To get started, where "." can be replaced with the path to `QG.mla`, from within Maple type:

```
libname := "./QG.mla", libname;
with(QuantumGraphs);
```

The module contains two functions:

`qgraphsa`	Determines whether a quantum graph coupled with a set of local vertex conditions gives rise to a self-adjoint extension of the operator.
`qginvariants`	Provides the same functionality as qgraphsa but does not check the vertex conditions.

Both functions return a boolean value, either *true* or *false* as to whether a self-adjoint extension is possible or determined by the given vertex conditions. If *false* is returned, the condition and the violating vertex are returned. If multiple vertices violate the condition, the first offending vertex is returned. The input for a graph Γ of n edges and p vertices is a Maple list

$$[A, V_1, \dots, V_p]\,,$$

where A is the list of operators $A = [A_1, \dots, A_n]$, and V_i is a list of lists $V_i = [\mathcal{F}, \mathcal{BC}]$. Here $\mathcal{F}$ is a list identifying the endpoint space present at the vertex V_i and $\mathcal{BC}$ is a list representing the homogeneous boundary conditions at the vertex. A sample input is provided in Appendix B.

3.3. Examples.

3.3.1. *Example 1.* We first consider the three interval system pictured in Figure 1 with the one dimensional Laplacian, $-\Delta$, acting upon each edge. Our algorithm correctly labeled the system as self-adjoint when either the Kirchhoff, δ, or δ' conditions were met.

When the conditions were not met, the algorithm correctly identified the system as being not self-adjoint and identified the offending vertex causing this to be the case. The input and output is provided in Appendix B.

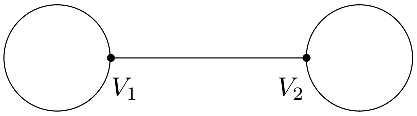

FIGURE 1. $-\Delta$ on three interval graph.

3.3.2. *Example 2.* We next consider the system pictured in Figure 2 under the conditions where f is continuous and the first derivatives at both endpoints of the loop sum to 0. Note that the left loop consists of $-\mathrm{i}\frac{d}{dx}$ and $\mathrm{i}\frac{d}{dx}$, the same operator with only a sign change. If both operators were of the same sign, the symplectic invariants would be $(1, 0)$ and $(0, 1)$, respectively. If this were the case, our algorithm would report that a self-adjoint extension is not possible.

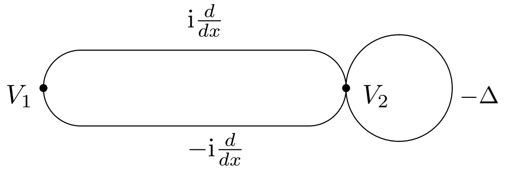

Figure 2. Three interval graph with multiple operators.

4. Conclusions

The ultimate result of this work was the development of a Maple application capable of validating the conditions of Theorem 2.18 for a given quantum graph. We hope that the application will be of benefit to those with interest in the field. Not only can our application serve as a validation step in problem solving, it can also be used in a more stand-alone nature for studying the effect that boundary conditions and graph structure have on self-adjointness. It would be of great use to develop a visual front-end for the application, including means of exploring the spectral properties for a given graph.

Acknowledgement. The authors would like to thank Thomas Krainer for giving advice to this project in the context of the Penn State–Göttingen Summer School 2010.

Appendix A. Maple Application

This appendix details the Maple application. The files are available online at http://math.aa.psu.edu/∼summerschool

```
QuantumGraphs := module ()

        local ModuleLoad, A, sf, phi, w, normalizeDiag,
                normalizeColumns, subscriptIndex, ns2matrix,
                splitMatrix, poly2taylor, dimTotal, diffOpOrder,
                checkRank, checkFormalSA, checkZero, genOpCoeffMatrix,
                generateT, generateH, generateVertexH, transform,
                checkLeadingCoeff;
                export qgraphsa, qginvariants;
        option package;

        ModuleLoad := proc ()
                print("Welcome to the QuantumGraphs package");
                print("Authors: Steven Coulter and Helene Dallmann,
                           Penn State / Gottingen Summer School 2010")
        end proc;

        # Generic Differential Operator
        A := proc (e, u, m, lambda)
                options operator, arrow;
                (m(e, 1))(x)*u+add((m(e, s+1))(x)*(diff(u,‘$‘(x, s))), s = 1 .. diffOpOrder(e, m))
        end proc;

        # Left Symplectic Form
        sf[l] := proc (e, m, u, v)
```

```
                options operator, arrow;
                int(A(e, expand(u), m, 0)*conjugate(expand(v)), x = 0.. 1/2)
                -(int(expand(u)*conjugate(A(e, expand(v), m, 0)), x = 0 .. 1/2)) end
proc;

        # Right Symplectic Form
        sf[r] := proc (e, m, u, v)
        options operator, arrow;
                int(A(e, expand(u), m,0)*conjugate(expand(v)), x = 1/2 .. 1)
                -(int(expand(u)*conjugate(A(e, expand(v),m, 0)), x = 1/2 .. 1))
        end proc;

        # Left Cutoff Function
        phi[l] := proc (n, x)
                options operator, arrow;
                x^(2*n)*(1/2-x)^(2*n)
        end proc;

        w[l] := proc (n, x)
                options operator, arrow;
                (int(phi[l](n, t), t = x .. 1/2))/(int(phi[l](n, t), t = 0 .. 1/2))
        end proc;

        # Right Cutoff Function
        phi[r] := proc (n, x)
                 options operator, arrow;
                 (1-x)^(2*n)*(x-1/2)^(2*n)
        endproc;

        w[r] := proc (n, x)
                options operator, arrow;
                (int(phi[r](n, t), t = 1/2 ..x))/(int(phi[r](n, t), t = 1/2 .. 1))
        end proc;

        # Normalize Diagonal Elements
        normalizeDiag := proc(m)::Matrix;
                local i, size, D; size := LinearAlgebra:-RowDimension(m);
                D := Matrix(size);
                for i to size do
                        D[i, i] := abs(1/sqrt(-I*m(i, i)))
                end do;
                return D
        end proc;

        # Normalize Columns
        normalizeColumns := proc (m)::Matrix;
                local i, size; size := LinearAlgebra:-RowDimension(m);
                for i to size do
                        m[1 .. size, i] := LinearAlgebra:-Normalize(LinearAlgebra:-Column(m, i), 2)
                end do
        end proc;

        # Get SubScript Index
        subscriptIndex := proc (list, j)::integer;
                return op(1, op(0, list[j]))
        end proc;

        # Join Nullspace Vectors into Matrix
        ns2matrix := proc (m)::Matrix;
                local NS, N, j;
                NS := LinearAlgebra:-NullSpace(m);
                N := NS[1];
                for j from 2 to nops(NS) do
                        N :=n‘<,>‘(‘<|>‘(N, NS[j]))
                end do;
                return N
        end proc;

        # Vertically Split Matrix
        splitMatrix := proc(m)::Matrix;
                local c, E, F;
                c := LinearAlgebra:-ColumnDimension(m);
        E := m[1 ..LinearAlgebra:-RowDimension(m), 1 .. (1/2)*c];
        F := m[1 ..LinearAlgebra:-RowDimension(m), (1/2)*c+1 .. c];
        return E, F
end proc;

# Find Total Dimension
dimTotal:= proc ()::integer;
        local i, d; d := 0;
        for i to numIntervals do
                d := d+n[i]
end do;
return d
end proc;

# Order of Differential Operator
diffOpOrder := proc (e, m)
        local opp, ord, i;
        opp :=convert(m(e, () .. ()), list);
        ord := 0;
        for i to nops(opp) do
                if opp[i] <> 0 then
                        ord := ord+1
                end if
        end do;
        return ord-1
end proc;

# Full Rank Test
checkRank := proc (m)
        return testeq(LinearAlgebra:-Rank(m),
                min(LinearAlgebra:-RowDimension(m),LinearAlgebra:-ColumnDimension(m)))
end proc;

# Formally Self Adjoint Test
checkFormalSA := proc (e, m)
        local n, adj, i, j;
        assume(x, ’real’);
        n := nops(LinearAlgebra[Row](m, e));
        adj :=(m(e, 1))(x)*u(x);
        for i from 2 to n do
                j := i-1;
                adj := adj+(-1)^j*(diff(conjugate((m(e, i))(x))*u(x), [‘$‘(x, j)]))
        end do;
        return testeq(adj, A(e, u(x), m, 0))
end proc;

# Ellipticity Test
checkLeadingCoeff := proc (e, m)
        local n, i, lead, value;
        n := nops(LinearAlgebra[Row](m, e));
        for i from n by -1 to 1 do
                if m(e, n) <> 0 then
                        lead := (m(e, n))(x)
                end if
        end do;
        value := evalf(minimize(abs(lead), x = 0 .. 1));
        return ‘not‘(testeq(value, 0))
end proc;

# Matrix Zero Check
checkZero := proc (m)
        return verify(LinearAlgebra:-ZeroMatrix(LinearAlgebra:-RowDimension(m)), m, Matrix)
end proc;

# Matrix of DiffOp Coeffs
genOpCoeffMatrix := proc (list)::Matrix;
        local i, j, finished, count, coeffList, nextCoeff, OpCoeffMatrix, adjust, inc;
        for i to nops(list) do
                finished := 0;
                j := 1;
                count := nops([coeffs(expand(list[i]))]);
                coeffList := [unapply(coeftayl(list[i], f = 0,0), x)];
                inc := nops([coeffs(coeftayl(list[i], f = 0, 0))]);
                if coeffList[1] <> (proc (x) options operator, arrow; 0 end proc) then
                        finished := finished+inc
                end if;
                while finished <> count do
                        nextCoeff := coeff(list[i], ((D@@j)(f))(x));
                        coeffList := [op(coeffList), unapply(nextCoeff, x)];
                        if nextCoeff <> 0 then
                                finished := finished+nops([coeffs(nextCoeff)])
                        end if;
                        j := j+1
                end do;
                adjust:= nops(coeffList)-LinearAlgebra:-ColumnDimension(OpCoeffMatrix);
                if 0 <= adjust then
                        OpCoeffMatrix := ‘<|>‘(‘<,>‘(‘<,>‘(‘<|>‘
                                (OpCoeffMatrix,LinearAlgebra:-ZeroMatrix
                                (LinearAlgebra:-RowDimension(OpCoeffMatrix), adjust))),
                                convert(coeffList, Matrix)))
                else
                        OpCoeffMatrix := ‘<|>‘(‘<,>‘(OpCoeffMatrix,‘<,>‘(‘<|>‘(
                        convert(coeffList, Matrix), Matrix(1, abs(adjust))))))
                end if
        end do;
        return OpCoeffMatrix
end proc;

# Taylor Matrix
generateT := proc (V, m)::Matrix;
        local T, j, k, n, count, numEndPointSpaces, coeffMatrixWidth;
        numEndPointSpaces :=nops(V[1]);
        coeffMatrixWidth := 0;
        for j to numEndPointSpaces do
                coeffMatrixWidth := coeffMatrixWidth+diffOpOrder(subscriptIndex(V[1], j), m)
        end do;
        T := Matrix(nops(V[2]), coeffMatrixWidth);
        count := 1;
        for j tobnumEndPointSpaces do
                for k from 0 to diffOpOrder(subscriptIndex(V[1], j), m)-1 do
                        for n to nops(V[2]) do
                                T[n, count] := coeff(V[2][n], ((D@@k)(op(0,V[1][j])))(op(1, V[1][j])))
                        end do;
                        count := count+1
                end do
        end do;
        return T
end proc;

# H Matrix
generateH := proc (e, side, m)::Matrix;
        local k, j, H, n;
        n := diffOpOrder(e, m);
        H := Matrix(n);
        for k to n do
                for j to n do
                        if side = 0 then
                                H[k, j] := sf[l](e, m, w[l](n, x)*x^(k-1)/
                                        factorial(k-1), w[l](n,x)*x^(j-1)/factorial(j-1))
                        else
                                H[k, j] := sf[r](e, m, w[r](n,x)*(x-1)^(k-1)/
                                        factorial(k-1), w[r](n, x)*(x-1)^(j-1)/factorial(j-1))
                        end if
                end do
        end do;
        return H
end proc;

# H Matrix for Entire Vertex
generateVertexH := proc (V, m)::Matrix;
        local H, M, i;
        H := Matrix();
        for i to nops(V[1]) do
                M := generateH(op(1, op(0,V[1][i])), op(V[1][i]), m);
                H := ‘<,>‘(‘<|>‘(‘<,>‘(H,Matrix(LinearAlgebra:-RowDimension(M),
                        LinearAlgebra:-ColumnDimension(H))),‘<,>‘(
                        Matrix(LinearAlgebra:-RowDimension(H), LinearAlgebra:-ColumnDimension(M)),M)))
        end do;
        return H
end proc;

# Transformation
transform := proc (m)::Matrix;
        local Q, P, M, E,p, n, size, i;
        global fail;
        E, Q := LinearAlgebra:-Eigenvectors(m);
        size :=LinearAlgebra:-RowDimension(Q);
        M := Matrix(size);
        p := 0;
        n := 0;
        for i to size do
                if 0 < Im(E(i)) then
                        M(1 .. size, p+1) := Q(1 .. size, i);
                        p := p+1
                elif Im(E(i)) < 0 then
                        M(1 .. size, size-n) := Q(1 .. size, i);
                        n := n+1
                end if
        end do;
        Q := M;
        normalizeColumns(Q);
        P :=bnormalizeDiag(LinearAlgebra:-HermitianTranspose(Q).m.Q);
        return P, Q, p, n
end proc;

# Self-Adjointness Test
qgraphsa := proc (list)
        local i, V, OCM, T, N, H, P, Q, p, q, M, E, F, X;
        for i from 2 to nops(list) do
                V[i-1] := list[i]
        end do;

        # Generate Coefficient Matrix,
        # Test for Formal Self-Adjointness
        OCM := genOpCoeffMatrix(list[1]);
        for i to nops(list[1]) do
                if checkFormalSA(i, OCM) = FAIL then
                        print(i-‘not formally self adjoint‘);
                        return false
                end if
        end do;

        # Loop over Vertices
        for i to nops(list)-1 do
                T := generateT(V[i], OCM);

                # Check Rank of Taylor Matrix
                if checkRank(T) = false then
                        print(i-‘not full rank‘);
                        return false
                end if;

                # Transform
                N := ns2matrix(T);
                H := generateVertexH(V[i], OCM);
                P, Q, p, q := transform(H);

                # Check Invariants
                if p-q <> 0 then
                        print(i-‘p not equal q‘);
                        return false
                end if;

                # Split
                M := convert(LinearAlgebra:-Transpose(N).(1/P).
                        (1/LinearAlgebra:-HermitianTranspose(Q)), Matrix);
                        E, F := splitMatrix(M);
                        X := simplify(E.LinearAlgebra:-HermitianTranspose(E)
                                -F.LinearAlgebra:-HermitianTranspose(F));

                        # Check Condition
                        if checkZero(X) = false then
                                print(i-‘vertex condition fail‘);
                                return false
                        end if
                end do;

                return true
        end proc;

        # Same as ggraphsa, but only check up to invariants
        qginvariants := proc (list)
                local i, V, OCM, T, N, H, P, Q, p, q, M, E, F, X;

                for i from 2 to nops(list) do
                        V[i-1] := list[i] end do;
                        OCM := genOpCoeffMatrix(list[1]);

                        for i to nops(list)-1 do
                                if checkFormalSA(i, OCM) = FAIL then
                                        print(i-‘not formally self adjoint‘);
                                        return false
                                end if;

                                H := generateVertexH(V[i], OCM);
                                P, Q, p, q := transform(H);
                                if p-q <> 0 then
                                        print(i-‘p not equal q‘);
                                        return false
                                end if
                        end do;
                return true
        end proc
end module

#Save Library
LibraryTools:-Save("/tmp/QG.mla")
libname := "/tmp/QG.mla", libname
savelib(QuantumGraphs)
```

Appendix B. Examples

This appendix details input/output for the example provided in the paper.

```
libname := "/tmp/QG.mla", libname;
with(QuantumGraphs);

#Example 1a: Laplace operator on a three edge, two vertex graph, meeting Kirchhoff conditions
B := [-((D@@2)(f))(x), -((D@@2)(f))(x), -((D@@2)(f))(x)];
V[1] := [[f[1](0), f[1](1), f[2](0)],
        [f[1](0)-f[1](1), f[1](1)-f[2](0), (D(f[1]))(0)-(D(f[1]))(1)+(D(f[2]))(0)]];
V[2] := [[f[3](0), f[3](1), f[2](1)],
        [f[3](0)-f[3](1), f[3](1)-f[2](1), (D(f[3]))(0)-(D(f[3]))(1)-(D(f[2]))(1)]];
qgraphsa([B, V[1], V[2]]);

                              true

#Examples  1b: Same configuration as previous examples with not meeting Kirchhoff conditions
B := [-((D@@2)(f))(x), -((D@@2)(f))(x), -((D@@2)(f))(x)];
V[1] := [[f[1](0), f[1](1), f[2](0)],
        [f[1](0)-f[1](1), 2*f[1](1)-f[2](0), (D(f[1]))(0)-(D(f[1]))(1)+(D(f[2]))(0)]];
V[2] := [[f[3](0), f[3](1), f[2](1)],
        [f[3](0)-f[3](1), f[3](1)-f[2](1), (D(f[3]))(0)-(D(f[3]))(1)+(D(f[2]))(1)]];
qgraphsa([B, V[1], V[2]]);

                   1 - vertex condition fail
```

```
                             false

#Examples  1c: Same configuration as previous examples, meeting  conditions
B := [-((D@@2)(f))(x), -((D@@2)(f))(x), -((D@@2)(f))(x)];
V[1] := [[f[1](0), f[1](1), f[2](0)],
        [f[1](0)-f[1](1), f[1](1)-f[2](0), (D(f[1]))(0)-(D(f[1]))(1)+(D(f[2]))(0)-f[1](0)]];
V[2] := [[f[3](0), f[3](1), f[2](1)],
        [f[3](0)-f[3](1), f[3](1)-f[2](1), (D(f[3]))(0)-(D(f[3]))(1)-(D(f[2]))(1)-f[3](0)]];
qgraphsa([B, V[1], V[2]]);

                              true

#Examples  1d: Same configuration as previous examples, meeting ’ conditions
B := [-((D@@2)(f))(x), -((D@@2)(f))(x), -((D@@2)(f))(x)];
V[1] := [[f[1](0), f[1](1), f[2](0)],
        [(D(f[1]))(0)-(D(f[1]))(1), (D(f[1]))(1)-(D(f[2]))(0), f[1](0)-f[1](1)+f[2](0)-(D(f[1]))(0)]];
V[2] := [[f[3](0), f[3](1), f[2](1)],
        [(D(f[3]))(0)-(D(f[3]))(1), (D(f[3]))(1)-(D(f[2]))(1), f[3](0)-f[3](1)-f[2](1)-(D(f[3]))(0)]];
qgraphsa([B, V[1], V[2]]);

                              true

#Example 2 a: Laplace operator on the loop, d/(dx) on the remaining two edges.
B := [I*((D@@1)(f))(x), -I*((D@@1)(f))(x), -((D@@2)(f))(x)];
V[1] := [[f[1](1), f[2](1), f[3](0), f[3](1)],
        [f[1](1)-f[2](1), f[3](0)-f[3](1), (D(f[3]))(0)-(D(f[3]))(1)]];
V[2] := [[f[1](0), f[2](0)], [f[1](0)-f[2](0)]];
qgraphsa([B, V[1], V[2]]);

                              true
```

References

[1] M.S. Birman and M.Z. Solomyak, *Spectral Theory of Self-Adjoint Operators in Hilbert Space*, D.Reidel Publishing Company, 1987.

[2] W.N. Everitt and L. Markus, *Boundary Value Problems and Symplectic Algebra for Ordinary Differential and Quasi-Differential Operators*, Mathematical Surveys and Monographs, vol. 61, American Mathematical Society, 1999.

[3] W.N. Everitt and L. Markus, *Multi-Interval Linear Ordinary Boundary Value Problems and Complex Symplectic Algebra*, Memoirs of the American Mathematical Society, vol. 715, American Mathematical Society, 2001.

[4] J.M. Harrison, *Quantum graphs with spin Hamiltonians*, in: *Analysis on Graphs and Its Applications*, pp. 261–277, Proceedings of Symposia in Pure Mathematics, vol. 77, American Mathematical Society, 2008.

[5] V. Kostrykin and R. Schrader, *Kirchhoff's rule for quantum wires*, J. Phys. A **32** (1999), 595–630.

[6] P. Kuchment, *Quantum graphs: an introduction and a brief survey*, in: *Analysis on Graphs and Its Applications*, pp. 291–312, Proceedings of Symposia in Pure Mathematics, vol. 77, American Mathematical Society, 2008.

[7] J. Weidmann, *Linear Operators in Hilbert Spaces.* Graduate Texts in Mathematics, Vol. 68, Springer-Verlag, New York/Heidelberg/Berlin 1980.

[8] J. Weidmann, *Spectral Theory of Ordinary Differential Operators.* Lecture Notes in Mathematics, Vol. 1258, Springer-Verlag, New York/Heidelberg/Berlin 1987.

Universität Göttingen
E-mail address: helene.dallmann@stud.uni-goettingen.de

The Pennsylvania State University
E-mail address: coulter@math.psu.edu