SIAM J. SCI. COMPUT.
Vol. 47, No. 2, pp. B428–B453

QGLAB: A MATLAB PACKAGE FOR COMPUTATIONS ON QUANTUM GRAPHS*

ROY H. GOODMAN†, GRACE CONTE‡, AND JEREMY L. MARZUOLA§

Abstract. We describe QGLAB, a new MATLAB package for analyzing partial differential equations on quantum graphs. The software is built on the existing, object-oriented MATLAB directed-graph class, inheriting its structure and adding additional easy-to-use features. The package allows one to construct a quantum graph and accurately compute the spectrum of elliptic operators, solutions to Poisson problems, the linear and nonlinear time evolution of a variety of PDEs, the continuation of branches of steady states (including locating and switching branches at bifurcations), and more. It overcomes the major challenge of discretizing quantum graphs—the enforcement of vertex conditions—using nonsquare differentiation matrices. It uses a unified framework to implement finite-difference and Chebyshev discretizations of differential operators on a quantum graph. For simplicity, the package overloads many built-in MATLAB functions to work on the class.

Key words. quantum graphs, differential equations, block operators

MSC codes. 65M70, 34B45, 35-02

DOI. 10.1137/23M1627729

See reproducibility of computational results at end of the article.

1. Introduction.

This paper introduces the main ideas used to build QGLAB, a software package written in MATLAB for computations on quantum graphs, and provides several examples of its use and accuracy [34]. The supplementary materials present more thorough operating instructions and additional computational examples.

Quantum graphs, networks of one-dimensional edges interacting via vertex conditions, appear in the literature going back many years. The modern study of the subject begins with an analysis of their spectral statistics in [41], whose authors coined the term "quantum graphs." The spectral theory and properties of quantum graph operators were further developed in [29], where quantum graphs were realized as the limits of quantum equations on thin wire-like domains; see also [30, 37] and the references therein. Quantum graphs provide effectively one-dimensional model equations that enable explicit calculations that serve as a backbone for representing geometric and spectral theoretic properties of more complicated higher-dimensional quantum models. For further introduction and history, we recommend [9, 11].

Numerical packages are essential to facilitate further study for several common reasons: making progress on larger-scale problems and those with nonlinearities and

*Submitted to the journal's Software, High-Performance Computing, and Computational Science and Engineering section January 2, 2024; accepted for publication (in revised form) November 13, 2024; published electronically March 31, 2025.
https://doi.org/10.1137/23M1627729
Funding: The first author acknowledges support from the NSF through NSF grant DMS-2206016. The second and third authors acknowledge support from the NSF through NSF CAREER grant DMS-1352353, NSF grant DMS-1909035, and NSF FRG grant DMS-2152289.

†Department of Mathematical Sciences, New Jersey Institute of Technology, Newark, NJ 07102 USA (goodman@njit.edu).
‡Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20723 USA (gconte23@unc.edu).
§Department of Mathematics, University of North Carolina, Chapel Hill, NC 27599 USA (marzuola@email.cunc.edu).

time dependence all depend on intuition built from numerical experimentation and visualization. This project provides high-level tools that allow users to quickly and easily set up, solve, and visualize the solutions to problems posed on quantum graphs. It overloads many built-in MATLAB commands for basic calculations and plotting.

The package's foundation is a quantum graph class built on the MATLAB directed-graph class. While we have striven to write a general-purpose software package for quantum graph computations, the direction of development has been guided by two classes of problems of research interest to its authors:

1. Computing bifurcation diagrams: standing waves of the nonlinear Schrödinger equation (NLS) occur along one-parameter families or *branches* rather than at isolated points. Such branches must be computed using continuation methods to understand the solutions' parameter dependence. Branches may cross, and the stability of solutions changes at isolated *bifurcation points.* The package can detect the most common bifurcations and switch branches; see [12, 35].
2. Spectral accuracy in space and high-order time-stepping: a planned future project is to compute time-periodic and time-relative-periodic orbits of the full time-dependent NLS on a compact quantum graph.

QGLAB grew from numerical studies in the authors' research [35, 39, 45]. Other works have developed numerical packages for quantum graphs. The most complete is *GraFiDi,* a Python-based package described in [14], which overlaps with QGLAB but has fewer features. Malenova built a small quantum graph package using *Chebfun* [44]. Others have studied finite-difference, finite-element, and Galerkin-based numerical methods without creating a software package for general use [3, 12, 18, 19].

QGLAB is distinguished by the breadth of problems it can handle: linear and nonlinear, stationary and time-dependent, constant and variable coefficient. It separates the solution of differential equations along a graph's edges from the constraint of solving the vertex conditions. This has allowed us to develop a language in which the numerical algorithms for solving problems on quantum graphs can be expressed in just a few lines, as demonstrated in numerous examples.

1.1. Defining a quantum graph. A quantum graph Γ consists of a directed metric graph, considered as a *complex of edges,* on which a function space and differential operators are defined. To be more specific, we define the graph $\Gamma = (\mathcal{V}, \mathcal{E})$ as a set of vertices $\mathcal{V} = \{\mathtt{v}_n,\, n = 1, \ldots, |\mathcal{V}|\}$ and a set of directed edges $\mathcal{E} = \{\mathtt{e}_m = (\mathtt{v}_i \to \mathtt{v}_j),\, m = 1, \ldots, |\mathcal{E}|\}$ and to each edge assign a positive length ℓ_m and impose upon the edge a coordinate x that increases from 0 to ℓ_m as the edge is traversed from the $\mathtt{v}_i$ to $\mathtt{v}_j$. In general, we may consider the lengths of some or all of the edges infinite, in which case that edge will be connected to a single vertex, but we defer this to later publications. Define the *degree* d_n of vertex $\mathtt{v}_n$ as the number of edges that include that vertex as an initial or final point, counting twice if an edge connects the vertex to itself.

A graph with $|\mathcal{V}| = 3$ vertices and $|\mathcal{E}| = 5$ edges is shown in Figure 1.1. The vertices have degrees $d_1 = 5$, $d_2 = 3$, and $d_3 = 4$. Under our definition, multiple edges may share the same initial and final vertex, as do edges $\mathtt{e}_1$ and $\mathtt{e}_2$.

A function $\Psi(x)$ defined on Γ is defined as a collection of functions on each of the edges $\Psi|_{\mathtt{e}_m} = \psi_m(x)$. The Laplace operator on the graph is defined by[1]

$$\triangle|_{\mathtt{e}_m} = \frac{\mathrm{d}^2}{\mathrm{d}x^2} - V_m(x) \quad \text{for} \quad 0 < x < \ell_m, \tag{1.1}$$

[1]With the inclusion of such a potential, the operator $-\triangle$ is more properly called a Schrödinger operator, but we abuse terminology for simplicity.

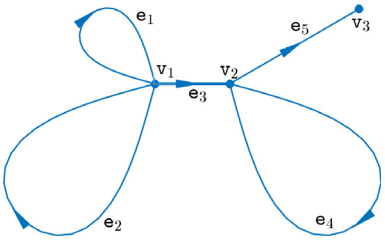


FIG. 1.1. *A directed graph with three vertices and five edges.*

subject to appropriate compatibility conditions at the vertices, given the presence of a potential $V(x)$ with $V(x)|_{\mathtt{e}_m} = V_m(x)$ which we usually set to zero. To define a Laplacian requires a function space. The graph and vertex conditions define a quantum graph, and we are most concerned with vertex conditions giving rise to a self-adjoint operator. The text [11] describes general criteria for self-adjoint vertex conditions. We discuss some of these here.

We assume the function is continuous at each vertex, yielding $d_n - 1$ conditions at vertex $\mathtt{v}_n$. Let $\mathcal{V}_n$ be the set of edges adjacent to this vertex, double counting self-directed edges. Continuity defines a function value at the vertex to be

$$\Psi(\mathtt{v}_n) \equiv \psi_i(\mathtt{v}_n) = \psi_j(\mathtt{v}_n) \,\forall \mathtt{e}_i, \mathtt{e}_j \in \mathcal{V}_n. \tag{1.2}$$

We define the weighted Robin–Kirchhoff vertex condition as

$$\sum_{\mathtt{e}_m \in \mathcal{V}_n} w_m \psi_m'(\mathtt{v}_n) + \alpha_n \Psi(\mathtt{v}_n) = 0, \tag{1.3}$$

where the derivative is, in all cases, taken in the direction *pointing away* from the vertex. In the case $w_m \equiv 1$, $\alpha_n \equiv 0$, this reduces to the Neumann–Kirchhoff vertex condition, which is the natural generalization of the Neumann boundary condition on a line segment. Interpreting the equation $\triangle \Psi = 0$ as describing a steady state of the heat equation, the Neumann–Kirchhoff vertex condition states that the net heat flux into the vertex vanishes. Associating nonunit weights w_m to the edges then specifies that the flux from each edge is proportional to its weight. Such weights define *balanced* star graphs in [39]. Setting $\alpha_n \neq 0$ generalizes a Robin vertex condition and may be interpreted as a delta function potential at the vertex. The Dirichlet condition is

$$\Psi(\mathtt{v}_n) = 0. \tag{1.4}$$

Setting the right-hand side of (1.3) or (1.4) to a value $\phi_n \neq 0$ defines nonhomogeneous vertex conditions.

We define norms and function spaces, e.g., $L^p(\Gamma)$,

$$\|\Psi\|^p_{L^p(\Gamma)} = \sum_{m=1}^{|\mathcal{E}|} w_m \|\psi_m\|^p_{L^p}, \tag{1.5}$$

and the L^2 inner product

$$\langle \Psi, \Phi \rangle = \sum_{m=1}^{|\mathcal{E}|} w_m \int_0^{\ell_m} \psi_m^*(x)\phi_m(x)\mathrm{d}x. \tag{1.6}$$

The weights w_m must appear in these definitions for the conservation laws for evolution equations below and to make the Laplace operator (1.1) self-adjoint. We define $H^1(\Gamma)$ as the space of square-integrable functions with square-integrable first derivatives. More subtly, we define $L^2(\Gamma)$, $H^1(\Gamma)$, and $H^2(\Gamma)$ to be the space of functions that are in each of these function spaces edgewise, but we define L^2_Γ to be the space $L^2(\Gamma)$ equipped with the inner product (1.6). Similarly, H^1_Γ is the space of functions in $H_1(\Gamma)$ satisfying the continuity condition (1.2), and H^2_Γ consists of functions in $H^2(\Gamma)$ satisfying both (1.2) and either the vertex condition (1.3) or (1.4).

1.2. The eigenvalue problem. The first natural question about the Laplacian operator defined on Γ is its spectrum and eigenfunctions. Such properties have been studied extensively; for a small subset of relevant works, see, for instance, [2, 6, 9, 10, 11, 38] and the references therein.

The compact quantum graph Laplacian has only a discrete spectrum, so we must compute the set of eigenvalues λ and eigenfunctions Ψ such that

$$\triangle\Psi = \lambda\Psi. \tag{1.7}$$

The spectrum is countable, is unbounded below, and has finitely many positive values. The nonpositive eigenvalues $\lambda = -k^2$ can be found by seeking the analytic solution

$$\psi_m(x) = a_m e^{ikx} + b_m e^{ik(\ell_m - x)}, \qquad m = 1, 2, \ldots, |\mathcal{E}|. \tag{1.8}$$

The vertex conditions form a homogeneous system of $2|\mathcal{E}|$ linear equations. Its solution requires the vanishing of the determinant of the associated matrix $\mathbf{S}(k)$, a function $\Sigma(k)$ called the *secular determinant* which can be normalized to take real values when $k \in \mathbb{R}$ [11]. The recent dissertation [21] shows this holds under the more general vertex conditions (1.3). In addition to QGLAB's many numerical features, it can symbolically compute the graph's real-valued secular determinant.

Two recent publications note numerically computing a determinant and finding its zeros have high complexity and low accuracy, so that finding the zeros of Σ is a poor method for computing the spectrum [18, 19]. Both suggest that finding the values k^* where the condition number of $\mathbf{S}(k)$ diverges is faster and more accurate; they subsequently find the eigenfunction as null vectors of $\mathbf{S}(k^*)$. Further, both suggest using the null vectors of $\mathbf{S}(k^*)$ as the basis for Galerkin methods to solve stationary and evolutionary PDE on metric graphs. This approach is ill-suited to nonlinear problems and those with edgewise-defined potentials or semi-infinite edges: nonlinearities are cumbersome to implement in Galerkin methods, eigenfunctions in the presence of potentials do not take the form (1.8), and graphs with infinite edges may not even have a point spectrum. The present work demonstrates numerical solutions to the first two problems. Extension to infinite edges is planned.

1.3. PDE on a quantum graph. Our primary motivating problem for building QGLAB is the NLS

$$i\frac{\partial\Psi}{\partial t} = \triangle\Psi + (\sigma+1)\,|\Psi|^{2\sigma}\,\Psi, \tag{1.9}$$

where $\sigma \geq 0$ and $\sigma = 1$ is the most commonly studied cubic case. We are especially interested in the stationary NLS obtained by assuming $\Psi(x,t) = e^{i\Lambda t}\Psi(x)$,

$$\mathcal{F}(\Psi,\Lambda) \equiv \Lambda\Psi + \triangle\Psi + (\sigma+1)\,|\Psi|^{2\sigma}\,\Psi = 0. \tag{1.10}$$

The evolution of (1.9) conserves both the L^2 norm defined in (1.5) and an energy

(1.11)
$$E(\Psi) = \|\Psi'\|^2_{L^2(\Gamma)} - \|\Psi\|^{2(\sigma+1)}_{L^{2(\sigma+1)}_\Gamma} + \sum_{\mathtt{e}_m \in \mathcal{E}} w_m \int_{\mathtt{e}_m} V_m(x)\,|\Psi_m(x)|^2\,\mathrm{d}x + \sum_{\mathtt{v}_n \in \mathcal{V}} \alpha_n\,|\Psi(\mathtt{v}_n)|^2\,,$$

where Ψ' is defined edge by edge. The NLS equation on the real line also conserves a momentum functional, but NLS on quantum graphs does not unless certain other restrictions to the weights and initial conditions hold; see [39].

Linear and nonlinear PDEs on quantum graphs are longstanding and active subjects. Many groups have studied (1.10) as surveyed in [16, 50]. The existence of ground states and the solution stability of stationary solutions are two important questions often studied from a variational perspective. There are too many works in this direction to properly do the subject justice, but see, for instance, [1, 15, 20, 22, 23, 55] and the references within to get a sense of the field. Others have studied the existence and stability of stationary states for Dirac and KdV equations [15, 53]. Recent works, including [12, 32, 35, 45, 51], use asymptotic and bifurcation-theoretical approaches to analyze the existence of multiple branches of solutions to (1.10). The book [47] gives an excellent introduction to time-dependent PDE on graphs. References including [39, 48, 54] analyze the time-dependent phenomena exhibited by Schrödinger, Dirac, and KdV equations on graphs. This short overview gives a flavor of the questions that can be posed on quantum graphs and the breadth of topics yet to be explored.

QGLAB has been explicitly written for PDE with Laplacian spatial derivative terms. In addition to the previously discussed NLS equation, these include the wave equation [50], the heat equation [8, 17], and their nonlinear cousins such as the nonlinear Klein–Gordon equations, including sine-Gordon [33, 43, 54, 56] and the Kolmogorov–Petrovsky–Piskunov equation [26], all of which can be defined on a quantum graph. QGLAB provides examples of solving all of these PDEs.

1.4. Organization of the paper. Section 2 discusses the numerical methods QGLAB uses to discretize and solve equations posed on quantum graphs. The longest part, subsection 2.1, discusses the overall framework of the discretization and its implementation using both finite-difference and Chebyshev approximations of derivatives and the implementation of the vertex conditions. We apply this framework to discretize eigenvalue problems in subsection 2.2, where we also discuss the symbolic calculation of the secular determinant for the general class of vertex conditions discussed. Subsection 2.3 describes the nonlinear solvers and continuation algorithms, while subsection 2.4 describes the implementation of time-steppers for evolution equations. Section 3 is devoted to the MATLAB implementation of the tools discussed in QGLAB, including a discussion of the MATLAB directed-graph class in section 3.1 and the QGLAB quantum graph class, which is built on top of this, in section 3.2. Section 3.3 discusses basic operations on class objects. In section 4, we consider three examples illustrating QGLAB in practice: a Poisson problem, an eigenvalue problem, and an initial-value problem for the cubic NLS equation. We summarize our contributions and give an outlook on potential future features and applications in section 5. The supplement contains extensive additional materials in two sections. The first, section SM1, is devoted to demonstrating both the implementation and efficacy of QGLAB on a variety of examples, including stationary problems—eigenvalue problems, the Poisson equation, and the computation and continuation of standing waves and evolutionary PDE problems. The second, section SM2, contains a complete listing of user-callable function definitions and explicit instructions for their use.

2. Numerical methods. This section discusses the numerical methods used to implement the quantum graph class and solve various problems. It briefly presents some examples described in detail in section 4 of the supplement.

2.1. Discretization and vertex conditions. QGLAB can perform tasks including solving for nonlinear standing waves and numerically integrating evolution equations on a quantum graph. Most important is discretizing the Laplace operator and solving the Laplace and Poisson equations. It provides two discretizations: centered differences and Chebyshev collocation. They can be used interchangeably, although the Chebyshev discretization is, by construction, more accurate.

The two discretizations are implemented using a common framework: the function $\Psi(x)$ is approximated on an *extended grid* $\mathbf{x}^{\text{ext}}$ with enough points to approximate both the PDE solution and the vertex conditions. In contrast, the discretized PDE is satisfied on a smaller *interior* grid $\mathbf{x}^{\text{int}}$ containing two fewer points per edge. Thus, the discrete Laplacian matrix is nonsquare, with $2|\mathcal{E}|$ more columns than rows, mapping from approximations on $\mathbf{x}^{\text{ext}}$ to approximations on $\mathbf{x}^{\text{int}}$. The vertex conditions are implemented as constraints, not incorporated directly into the discretized Laplacian matrix. This choice has several attractive features described below.

Driscoll and Hale introduced nonsquare differentiation matrices using rectangular discretization matrices for use in the Chebfun package [24, 25]. Aurentz and Trefethen have written an excellent review, developing the theory for the *block operators* that implement these ideas via a sequence of well-chosen examples [5].

2.1.1. Finite-difference discretization. The finite-difference method is implemented using second-order centered differences with the boundary conditions enforced at so-called ghost points, as discussed, for example, in the textbook [28, sect. 4.2.2]. We review this technique for discretizing the two-point boundary value problem

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} - V(x)u(x) = f(x), 0 < x < \ell,\ u'(0) + \alpha_0 u(0) = \phi_0,\ -u'(\ell) + \alpha_\ell u(\ell) = \phi_\ell \tag{2.1}$$

and then discuss the straightforward extension to quantum graphs. The sign on the u' term in the boundary conditions is chosen to agree with our quantum graph convention that all derivatives are taken in the direction pointing away from a vertex of the quantum graph in the definition of vertex conditions. Given a discretization length $h = \frac{\ell}{N}$, place points on a grid offset by half a step size $x_k = (k - \frac{1}{2})h$ for $0 \le k \le N+1$ as shown in Figure 2.1. The first and last points lie outside the interval of interest, and the endpoints of the desired interval do not appear on the list of points. Letting u_k approximate $u(x_k)$, the discretized equation at the interior points is then

$$u''(x_k) - V(x_k)u(x_k) \approx \frac{u_{k-1} - 2u_k + u_{k+1}}{h^2} = f_k = f(x_k) \quad \text{for} \quad k = 1, \dots, N,$$

up to an error of $\mathcal{O}h^2$. The value of $u(0)$ and $u'(0)$ may be approximated to second order via suitable linear combinations of $u(\pm h/2)$, yielding a second-order approximation of the boundary condition (2.1),

$$\left(\frac{\alpha_0}{2} - \frac{1}{h}\right)u_0 + \left(\frac{\alpha_0}{2} + \frac{1}{h}\right)u_1 = \phi_0, \tag{2.2}$$

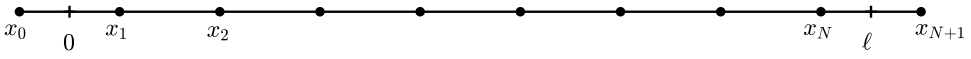


FIG. 2.1. *Discretization of the interval* $[0, \ell]$ *using ghost points.*

and a similar approximation for the right boundary condition. In the case of a Dirichlet boundary condition at $x = 0$, this is replaced by $\frac{1}{2}(u_0 + u_1) = \phi_0$.

Assembling the equations for the second derivatives and the boundary conditions into matrix-vector form, we define the vectors

$$\mathbf{u} = (u_0, u_1, \dots, u_{N+1})^{\mathsf{T}}, \mathbf{f} = (f_0, f_1, \dots, f_{N+1})^{\mathsf{T}}, \quad \text{and} \quad \boldsymbol{\phi} = (\phi_0, \phi_\ell)^{\mathsf{T}}, \tag{2.3}$$

as well as the $N \times (N+2)$ *interior projection matrix*

$$\mathbf{P}_{\text{int}} = \begin{pmatrix} \mathbf{0}_{N\times 1} & \mathbf{I}_N & \mathbf{0}_{N\times 1} \end{pmatrix}, \tag{2.4}$$

the $N \times (N+2)$ *interior Laplacian matrix*

$$\mathbf{L}_{\text{int}} = \frac{1}{h^2} \begin{pmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \end{pmatrix} - \mathbf{P}_{\text{int}} \cdot \operatorname{diag} V(x_k), \tag{2.5}$$

and the $2 \times (N+2)$ *boundary condition matrix*

$$\mathbf{M}_{\text{BC}} = \begin{pmatrix} \left(\frac{\alpha_0}{2} - \frac{1}{h}\right) & \left(\frac{\alpha_0}{2} + \frac{1}{h}\right) & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \left(\frac{\alpha_\ell}{2} + \frac{1}{h}\right) & \left(\frac{\alpha_\ell}{2} - \frac{1}{h}\right) \end{pmatrix}, \tag{2.6}$$

where $\mathbf{I}_N$ is the N-dimensional identity matrix and $\mathbf{0}_{M\times N}$ is a matrix of size $(M \times N)$ of all zeros. The matrices $\mathbf{L}_{\text{int}}$ and $\mathbf{P}_{\text{int}}$ are linear maps from approximation of u on the *extended grid* $\mathbf{x}^{\text{ext}} = \{x_0, \dots, x_{N+1}\}$ to its approximation on the *interior grid* $\mathbf{x}^{\text{int}} = \{x_1, \dots, x_N\}$. With this, we discretize the differential equation as

$$\mathbf{L}_{\text{int}}\mathbf{u} = \mathbf{P}_{\text{int}}\mathbf{f}, \tag{2.7}$$

and the boundary conditions as

$$\mathbf{M}_{\text{BC}}\mathbf{u} = \boldsymbol{\phi}. \tag{2.8}$$

We can then combine these into a single system of $N+2$ equations in $N+2$ unknowns

$$\left(\frac{\mathbf{L}_{\text{int}}}{\mathbf{M}_{\text{BC}}}\right)\mathbf{u} = \left(\frac{\mathbf{P}_{\text{int}}}{\mathbf{0}_{2\times(N+2)}}\right)\mathbf{f} + \left(\frac{\mathbf{0}_{N\times 2}}{\mathbf{I}_2}\right)\boldsymbol{\phi}. \tag{2.9}$$

We make two brief remarks on this approach. First, it is more common to solve the discretized boundary condition (2.8) for u_0 and u_{N+1} [19]. Inserting these values into (2.7) yields a reduced system with N unknowns. We leave the system in form (2.9) for two reasons: first, it makes implementing nonhomogeneous boundary conditions very easy, and the other is that it makes the approach more similar to how we handle boundary conditions using Chebyshev discretization. Second, the null space of the matrix $\mathbf{L}_{\text{int}}$ mimics that of the second derivative: it consists of vectors $\mathbf{v}$ with $v_n = an + b$ and is two-dimensional.

This scheme extends easily to the Poisson problem on the quantum graph,

$$\triangle\Psi(x) = f(x), \quad \text{i.e.,} \quad \psi_m''(x) - V_m(x)\psi_m(x) = f_m(x), \mathtt{e}_m, 1 \le m \le |\mathcal{E}| \tag{2.10a}$$

$$\psi_i(\mathtt{v}_n) = \psi_j(\mathtt{v}_n)\, \forall \mathtt{e}_i, \mathtt{e}_j \in \mathcal{V}_n, \quad \text{i.e., continuity,} \tag{2.10b}$$

$$\sum_{\mathtt{e}_m \in \mathcal{V}_n} w_m \psi_m'(\mathtt{v}_n) + \alpha_n \Psi(\mathtt{v}_n) = \phi_n \quad \text{or} \quad \Psi(\mathtt{v}_n) = \phi_n, 1 \le n \le |\mathcal{V}|\,. \tag{2.10c}$$

We discretize each edge $\mathtt{e}_m$ with the ghost-point formulation, with $N_m + 2$ discretization points defining $\mathbf{x}_m^{\text{ext}}$, and a mesh size $h_m = \ell_m/N_m$, generating $(N_m) \times N_m + 2$ matrices $\mathbf{L}_{\text{int}}^{(m)}$ and $\mathbf{P}_{\text{int}}^{(m)}$ of the same forms as matrices (2.5) and (2.4). Thus, letting $N_{\text{int}} = \sum_{m=1}^{|\mathcal{E}|} N_m$ and $N_{\text{ext}} = N_{\text{int}} + 2|\mathcal{E}|$, this results in N_{ext} unknowns arranged as

$$\boldsymbol{\psi} = \begin{pmatrix} \boldsymbol{\psi}^{(1)} \\ \vdots \\ \boldsymbol{\psi}^{(|\mathcal{E}|)} \end{pmatrix}, \quad \text{where} \quad \boldsymbol{\psi}^{(m)} = \begin{pmatrix} \boldsymbol{\psi}_0^{(m)} \\ \vdots \\ \boldsymbol{\psi}_{N_m+1}^{(m)} \end{pmatrix}. \tag{2.11}$$

The vector $\mathbf{f}$ is assigned similarly, and the vector of nonhomogeneous boundary terms is $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{|\mathcal{V}|})^{\mathsf{T}}$. Enforcing the continuity condition (2.10b) at the vertex $\mathtt{v}_n$ requires $(d_n - 1)$ rows and the Robin–Kirchhoff condition (2.10c) at vertex $\mathtt{v}_n$ involves the $2d_n$ adjacent discretization points in one row. The derivative and function values at the vertex are approximated to second order using a straightforward generalization of the reasoning leading to (2.2) and (2.6). Altogether, these form a matrix $\mathbf{M}_{\text{VC}}^{(n)}$ of dimension $(2d_n) \times N_{\text{ext}}$. We let

$$\mathbf{L}_{\text{VC}} = \left(\frac{\mathbf{L}_{\text{int}}}{\mathbf{M}_{\text{VC}}}\right) = \left(\begin{array}{ccc} \mathbf{L}_{\text{int}}^{(1)} & & \\ & \ddots & \\ & & \mathbf{L}_{\text{int}}^{(|\mathcal{E}|)} \\ \hline & \mathbf{M}_{\text{VC}}^{(1)} & \\ & \vdots & \\ & \mathbf{M}_{\text{VC}}^{(|\mathcal{V}|)} & \end{array}\right), \tag{2.12}$$

and

$$\mathbf{P}_0 = \left(\frac{\mathbf{P}_{\text{int}}}{\mathbf{0}_{2|\mathcal{E}| \times N_{\text{ext}}}}\right) = \left(\begin{array}{ccc} \mathbf{P}_{\text{int}}^{(1)} & & \\ & \ddots & \\ & & \mathbf{P}_{\text{int}}^{(|\mathcal{E}|)} \\ \hline & \mathbf{0}_{2|\mathcal{E}| \times (N_{\text{ext}})} & \end{array}\right). \tag{2.13}$$

We define the *nonhomogeneity matrix* $\mathbf{M}_{\text{NH}}$ in two steps. First define a matrix $\mathbf{M}$ of size $2|\mathcal{E}| \times |\mathcal{V}|$ such that

$$\mathbf{M}(i,j) = \begin{cases} 1 & \text{if the } j\text{th Neumann}-\text{Kirchhoff condition is enforced by row } i \text{ of } \mathbf{M}, \\ 0 & \text{otherwise,} \end{cases}$$

and then define

$$\mathbf{M}_{\text{NH}} = \left(\frac{\mathbf{0}_{N_{\text{int}}, |\mathcal{V}|}}{\mathbf{M}}\right). \tag{2.14}$$

This assigns each entry of the nonhomogeneous term $\boldsymbol{\phi}$ to the row of the system enforcing the vertex conditions. The discretization of system (2.10) can now be represented as

$$\mathbf{L}_{\text{VC}}\boldsymbol{\psi} = \mathbf{P}_0\mathbf{f} + \mathbf{M}_{\text{NH}}\boldsymbol{\phi}. \tag{2.15}$$

The use of this discretization is demonstrated in section 4.1.

We introduce some notation to simplify our discussion of the numerical problems addressed below. The matrices $\mathbf{P}_{\text{int}}$ and $\mathbf{L}_{\text{int}}$, of dimension $N_{\text{int}} \times N_{\text{ext}}$, represent linear maps from the function space $\mathbb{F}^{\text{ext}}$, of functions defined on the extended grid $\mathbf{x}^{\text{ext}}$, to the function space $\mathbb{F}^{\text{int}}$, of functions defined on the interior grid $\mathbf{x}^{\text{int}}$. Of course, $\mathbb{F}^{\text{ext}} = \mathbb{R}^N_{\text{ext}}$ and $\mathbb{F}^{\text{int}} = \mathbb{R}^N_{\text{int}}$, but thinking of these spaces merely as high-dimensional Euclidean spaces neglects the meaning to which we have assigned the elements of each space. Further, we denote by $\mathbb{F}^{\text{ext}}_{\boldsymbol{\phi}}$ the set of functions in $\mathbb{F}^{\text{ext}}$ which, in addition, satisfy the discretized boundary conditions represented by the final $2|\mathcal{E}|$ rows of system (2.15). Note that if $\boldsymbol{\phi} \neq \mathbf{0}$, i.e., for nonhomogeneous vertex conditions, then $\mathbb{F}^{\text{ext}}_{\boldsymbol{\phi}}$ is an affine space of dimension N_{int}, while for $\boldsymbol{\phi} = \mathbf{0}$, $\mathbb{F}^{\text{ext}}_{\mathbf{0}}$ is a linear vector space of dimension N_{int}. Thus the first N_{int} rows of (2.15) use the points from $\mathbf{x}^{\text{ext}}$ to approximately evaluate the underlying Laplace equations at the points in $\mathbf{x}^{\text{int}}$, while the remaining $2|\mathcal{E}|$ rows ensure that the solution lies on $\mathbb{F}^{\text{ext}}_{\boldsymbol{\phi}}$.

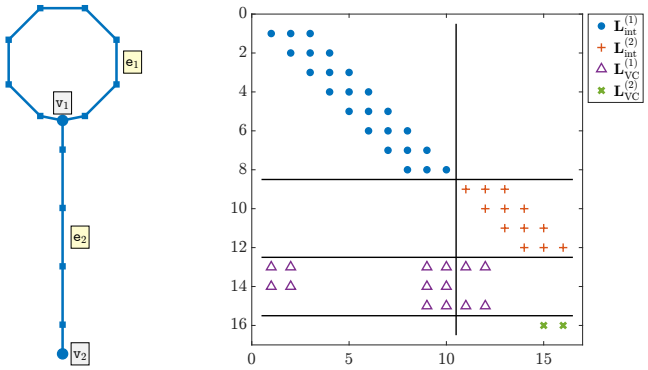
In what follows, we will apply similar reasoning to discretize other problems on the quantum graph. As above, we apply the differential equations at the interior points and supplement these equations with $2|\mathcal{E}|$ additional equations representing the vertex conditions, which suffice to specify a unique solution. In addition to the matrices $\mathbf{L}_{\text{VC}}$ and $\mathbf{P}_0$, we will use the matrices

$$\mathbf{L}_0 = \left(\frac{\mathbf{L}_{\text{int}}}{\mathbf{0}_{2|\mathcal{E}| \times N_{\text{ext}}}} \right) \quad \text{and} \quad \mathbf{P}_{\text{VC}} = \left(\frac{\mathbf{P}_{\text{int}}}{\mathbf{M}_{\text{VC}}} \right) \tag{2.16}$$

to develop time-stepping schemes for evolution equations in section 2.4.

Our choice of this ghost-point discretization stems from an effort to preserve the Laplacian's self-adjointness and, consequently, the realness of its eigenvalues postdiscretization. Ghost points have an important advantage over the standard on-point discretization for enforcing the boundary conditions in the BVP (2.1). The simplest second-order discretization uses a nonstaggered grid. It approximates the boundary condition at $x = 0$ to second-order accuracy at the boundary by applying a one-sided difference to $\frac{\mathrm{d}u}{\mathrm{d}x}|_{x=0}$ using the values u_0, u_1, and u_2, and similarly at $x = 1$. Solving the two discretized boundary conditions eliminates the values of u_0 and u_{N+1} from the system, leaving a system of N unknowns, but with an asymmetric finite-difference matrix, even though the differential operator which it approximates is self-adjoint. The ghost-point discretization, by contrast, preserves self-adjointness (after the two ghost points are first eliminated from the system).

This changes slightly when we extend this construction to the quantum graph Poisson problem (2.10). If all edges are discretized with the same step size h, the discretization matrix constructed above is symmetric. However, choosing all discretization lengths to be equal may be impossible or inconvenient. In that case, we may measure the magnitude of the asymmetry by considering the largest element, in absolute value, of the asymmetric part $\frac{1}{2}(\mathbf{A} - \mathbf{A}^{\mathsf{T}})$. Assume that edge $\mathtt{e}_m$ is discretized with a step size $h_m = h + \delta_m$ with $\delta_m = O(\delta) \ll h$. Then using one-sided centered differences introduces terms of $O(\frac{1}{h^2})$ into the asymmetric part of $\mathbf{A}$. By contrast, using ghost points introduces terms of $O(\frac{\delta}{h^2})$. Therefore, if all the discretization lengths h_m are roughly equal, the nonsymmetric part of $\mathbf{A}$ will be significantly smaller so that the matrix $\mathbf{A}$ is "more symmetric." Since $\mathbf{A}$ is not symmetric, we cannot guarantee that all of our eigenvalues are purely real, as they are for the underlying differential operator. However, by an informed choice of the discretization sizes, we may minimize the effects of the asymmetry



(a) (b)

FIG. 2.2. (a) *The lasso graph, showing interior discretization points.* (b) *The structure of the nonzero entries in the matrix* $\mathbf{L}_{\mathrm{VC}}$*, the Laplacian matrix extended with vertex conditions.*

An example discretization. We display the structure of the discretized Laplacian matrix of a lasso graph with two vertices and two edges in Figure 2.2. The edge $\mathtt{e}_1$ points from $\mathtt{v}_1$ to $\mathtt{v}_2$ and the edge $\mathtt{e}_2$ points from $\mathtt{v}_2$ to itself. The discretization has $N_1 = 4$ and $N_2 = 8$ points. The figure shows the interior points of the discretization and the vertices but not the ghost points. The figure also shows the nonzero structure of the matrix $\mathbf{L}_{\mathrm{VC}}$. The 16×2 matrix $\mathbf{M}_{\mathrm{NH}}$ is nonzero at $(13, 1)$ and $(14, 2)$.

2.1.2. Chebyshev discretization. To achieve spectral accuracy, QGLAB allows discretization using *rectangular collocation,* a method based on Chebyshev polynomials due to Driscoll and Hale [24], with further implementation details described in [57]. Enforcing nontrivial boundary conditions based on a nonsquare differentiation matrix with this method is especially simple. To introduce the idea, we again consider the Robin problem on a line segment defined in (2.1). The method uses two separate grids, as shown in Figure 2.3. The exterior grid $\mathbf{x}^{\mathrm{ext}}$ is given by the $N+2$ (increasing) Chebyshev points of the second kind,

$$x_k^{\mathrm{ext}} = \frac{\ell}{2}\left(1 - \cos\left(\frac{k\pi}{N+1}\right)\right), \qquad k = 0, 1, \ldots, N, N+1. \tag{2.17}$$

We adapt the notation of (2.3) to define the vectors $\mathbf{u}$, $\mathbf{f}$, and $\boldsymbol{\phi}$ on the discretization points defined in (2.17). The main observation motivating rectangular collocation is that applying a second derivative matrix defined over a finite space of polynomials should reduce the order of that space by two, naturally leading to matrices of size $N \times (N+2)$. This is realized by first operating on the vector $\mathbf{u}$ with the standard $(N+2) \times (N+2)$ Chebyshev derivative matrix $\mathbf{D}^2$ and then resampling these polynomials onto the interior grid $\mathbf{x}^{\mathrm{int}}$ of first-kind Chebyshev points,

$$x_k^{\mathrm{int}} = \frac{\ell}{2}\left(1 - \cos\left(\frac{(2k-1)\pi}{2N}\right)\right), \qquad k = 1, 2, \ldots, N. \tag{2.18}$$

As in the finite-difference discretization above, we define exterior and interior grids $\mathbf{x}^{\mathrm{ext}}$ and $\mathbf{x}^{\mathrm{int}}$. The approximate solutions are defined on $\mathbf{x}^{\mathrm{ext}}$, but derivatives, and thus approximations to the differential equations, are evaluated at the points of $\mathbf{x}^{\mathrm{int}}$. By contrast, the two grids are disjoint sets and the first and last points of the exterior grid $\mathbf{x}^{\mathrm{ext}}$ are the endpoints of the interval, not the ghost points.

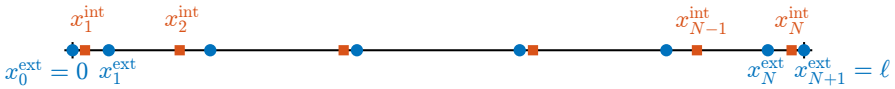


FIG. 2.3. *Discretization of the interval* $[0, \ell]$ *using a grid* $\mathbf{x}^{\text{ext}}$ *of Chebyshev points of the second kind (in blue) and a grid* $\mathbf{x}^{\text{int}}$ *first-kind Chebyshev points (in red). (Color figures are available online.)*

Resampling is a linear operation and is represented by an $N \times (N+2)$-dimensional *barycentric resampling matrix* $\mathbf{P}_{\text{int}}$ whose construction uses the barycentric interpolation formula proposed in [13]. Given the set of points $\mathbf{x}^{\text{ext}} = \{x_k^{\text{ext}}\}_{k=0}^{N+1}$, the barycentric weights are

$$\tilde{w}_k = \prod_{\substack{l=0 \\ l \neq k}}^{N+1} (x_k^{\text{ext}} + x_l^{\text{ext}})^{-1}, \qquad k = 0, \dots, N+1. \tag{2.19}$$

These are used to construct a unique interpolating polynomial $p_{N+1}(x)$ which interpolates the set of data points $\{(x_k^{\text{ext}}, f_k)\}_{k=0}^{N+1}$. The polynomial is evaluated at both $\{x_k^{\text{ext}}\}_{k=0}^{N+1}$ and $\{x_k^{\text{int}}\}_{k=1}^{N}$ so that the barycentric resampling matrix is given by

$$(\mathbf{P}_{\text{int}})_{j,k} = \begin{cases} \dfrac{\tilde{w}_k}{x_j^{\text{int}} - x_k^{\text{ext}}} \left(\displaystyle\sum_{l=0}^{N+1} \frac{\tilde{w}_l}{x_j^{\text{int}} - x_l^{\text{ext}}} \right)^{-1}, & x_j^{\text{int}} \neq x_k^{\text{ext}}, \\ 1, & x_j^{\text{int}} = x_k^{\text{ext}}, \end{cases} \tag{2.20}$$

and satisfies

$$p_{N+1}(\mathbf{x}^{\text{int}}) = \mathbf{P}_{\text{int}}\, p_{N+1}(\mathbf{x}^{\text{ext}}), \tag{2.21}$$

i.e., the barycentric resampling matrix maps a polynomial's values at the gridpoints $\mathbf{x}^{\text{ext}}$ to its values at the gridpoints $\mathbf{x}^{\text{int}}$. Driscoll and Hale generalize this in [24].

Putting this all together, the product

$$\mathbf{L}_{\text{int}} = \mathbf{P}_{\text{int}} \mathbf{D}^2 \tag{2.22}$$

defines an $N \times (N+2)$ differentiation matrix. The right-hand side of the differential equation (2.1) must be resampled to the same grid, so the differential equation is discretized by the N equations $\mathbf{L}_{\text{int}}\mathbf{u} = \mathbf{P}_{\text{int}}\boldsymbol{\phi}$, leaving two equations to define the boundary conditions (2.1). These may be compactly rewritten as

$$\mathbf{M}_{\text{BC}}\mathbf{u} = \begin{pmatrix} \phi_0 \\ \phi_L \end{pmatrix},$$

where $\mathbf{M}_{\text{BC}}$ is a matrix of size $2 \times (N+2)$ conveniently expressed using unit vectors

$$\mathbf{M}_{\text{BC}} = \begin{pmatrix} \mathbf{e}_1^{\mathsf{T}}\mathbf{D} + \alpha_0 \mathbf{e}_1^{\mathsf{T}} \\ -\mathbf{e}_{N+2}^{\mathsf{T}}\mathbf{D} + \alpha_L \mathbf{e}_{N+2}^{\mathsf{T}} \end{pmatrix}. \tag{2.23}$$

The matrices $\mathbf{L}_{\text{int}}$ and $\mathbf{P}_{\text{int}}$ are blockwise dense, in contrast to the banded matrices arising in the uniform discretization, defined in (2.6). We have constructed all the necessary elements to reinterpret (2.9) as a spectral collocation of (2.1).

Extending this argument from the ODE boundary problem proceeds as for the centered-difference approximation. Defining the matrices $\mathbf{L}_{\mathrm{VC}}$, $\mathbf{P}_0$, and $\mathbf{M}_{\mathrm{NH}}$ in (2.15) is straightforward once we construct the submatrix $\mathbf{M}_{\mathrm{VC}}^{(n)}$ defining the discretized vertex condition (2.12) and (2.13), extending the construction in (2.23). Figure 2.4 shows a coarse discretization of the example shown in Figure 2.2, in which the blocks defining the second derivative are dense, as are the rows defining the Robin–Kirchhoff vertex condition, whereas the rows enforcing continuity contain only two nonzero entries.

Two of the authors used this method to solve the Laplacian eigenproblem on an interval perturbed by several delta function potentials in [7].

2.2. Numerical and symbolic eigenproblems. Following the steps used above to discretize the Poisson problem, the eigenvalue problem (1.7) becomes

$$\mathbf{L}_{\mathrm{VC}}\mathbf{u} = \lambda \mathbf{P}_0 \mathbf{u} \tag{2.24}$$

in both the finite-difference and Chebyshev discretizations. Since $\mathbf{P}_0$ is not the identity matrix and is singular, this is a *generalized eigenvalue problem* which can be solved using `eigs` in MATLAB. QGLAB has overloaded the `eigs` command so that `[d,v]=G.eigs(m)` returns the m eigenvalues of the smallest absolute value. Figure 2.5 shows the eigenfunctions of the four smallest eigenvalues of the Laplacian on a Y-shaped graph with Dirichlet conditions at the ends of the two shorter edges. The example is described further in section 4.2. QGLAB's plotting features are described in section 3.3.1.

The zeros k_n of the secular determinant function $\Sigma(k)$, described in section 1.2, correspond to eigenvalues $\lambda_n = -k_n^2$ of the Laplacian operator. The QGLAB func-

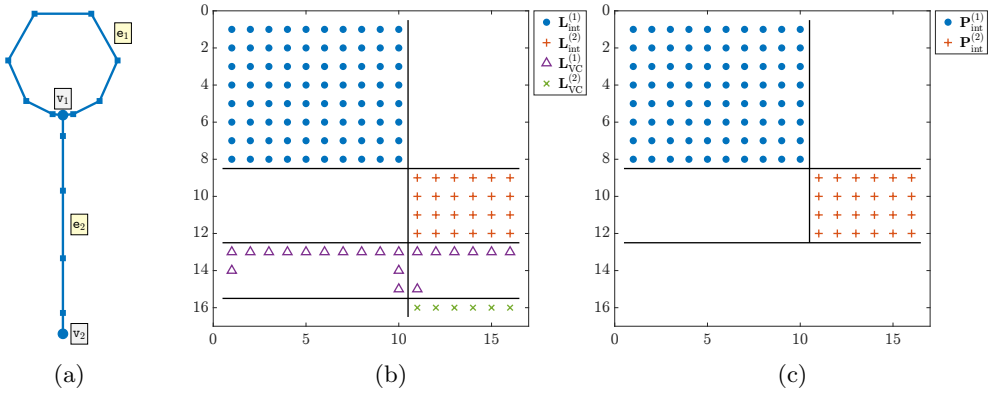


FIG. 2.4. (a) *The lasso graph, shown with interior discretization points in the Chebyshev discretization.* (b) *The structure of the nonzero entries in* $\mathbf{L}_{\mathrm{VC}}$*, the Laplacian matrix extended with vertex conditions.* (c) *The structure of* $\mathbf{P}_0$*, the barycentric resampling matrix extended with zeros.*

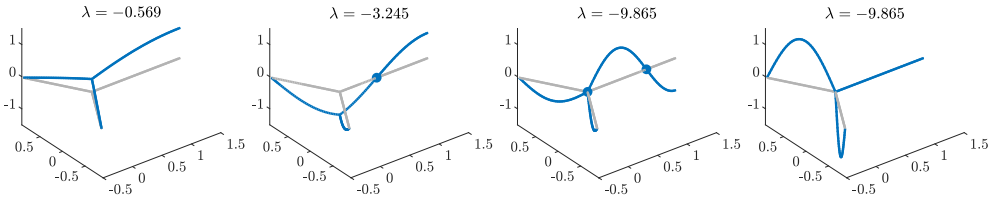


FIG. 2.5. *Four eigenfunctions of a Y-shaped quantum graph.*

tion `secularDet` uses the MATLAB Symbolic Math Toolbox to construct $\Sigma(k)$ for the boundary conditions (1.3) with $w_j \equiv 1$ and for Dirichlet boundary conditions and simplify it for typesetting and plotting. This is described in the dissertation [21]. It provides a check on the numerical calculation of eigenvalues for the discretized problem.

2.3. Nonlinear solvers, continuation, and bifurcation algorithms. After discretizing the spatial derivatives, we use QGLAB to construct the Newton–Raphson method to find standing wave solutions of the stationary NLS (1.10) with $\sigma = 1$. Fixing Λ, it proceeds by the iteration $\Psi_{n+1} = \Psi_n + \delta$ where $\delta \in \mathbb{F}_0^{\text{ext}}$ solves $D\mathcal{F}(\Psi_n, \Lambda) \cdot \delta = -\mathcal{F}(\Psi_n, \Lambda)$. Enforcing the vertex conditions gives an iteration of the form

$$\left(\mathbf{L}_{\text{VC}} + \mathbf{P}_0 \left(6\,\text{diag}(\Psi_n^2) + \Lambda \mathbf{I}\right)\right) \delta = -\mathbf{L}_0 \Psi_n - \mathbf{P}_0 \left(\Lambda \Psi_n + 2\,\text{diag}(\Psi_n^3)\right). \tag{2.25}$$

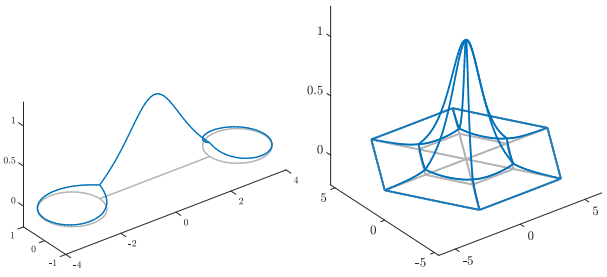
Two examples of such solutions are shown in Figure 2.6 and section 4.3.

Solutions to (1.10) do not occur at isolated points but along one-parameter families that, away from singularities, can be parameterized by the frequency Λ. Pseudoarclength continuation provides a way to follow this family as it traces a smooth path. It is due originally to Keller [40] and is well summarized in the textbook of Nayfeh and Balachandran [49], who cite many additional contributors to the method. The method allows curves to be continued around fold singularities, and other techniques can detect branch points, which encompass both pitchfork and transcritical bifurcations, and initiate new families of solutions branching off from the computed branch; see also Govaerts [36]. QGLAB's pseudoarclength continuation was first used to calculate bifurcation diagrams in [12].

QGLAB's contribution here is not a novel numerical method but the integration of pseudoarclength continuations into quantum graph software. Failure to use these methods has led to some seemingly impossible phenomena in the quantum graph literature: a solution branch that appears to end abruptly in [45], subsequently fixed in [35], and solutions with different symmetries seeming to lie on a single branch in [14], fixed in the example below.

The main step of the continuation algorithm is a Newton step like (2.25), in which the parameter Λ is left unknown. A predictor-corrector algorithm introduces one more equation to close the system. QGLAB is novel among quantum graph software because it implements continuation algorithms and because the operator language developed in section 2.1 allows a compact representation of the equations that arise.

An example computation demonstrates QGLAB's capabilities. Section SM1.1.2 of the supplement contains further examples, but these computations involve too many



(a) (b)

FIG. 2.6. (a) *A standing wave of cubic NLS on a dumbbell graph with* $\Lambda = -1$. (b) *A standing wave on a spiderweb graph with* $\Lambda = -1$.

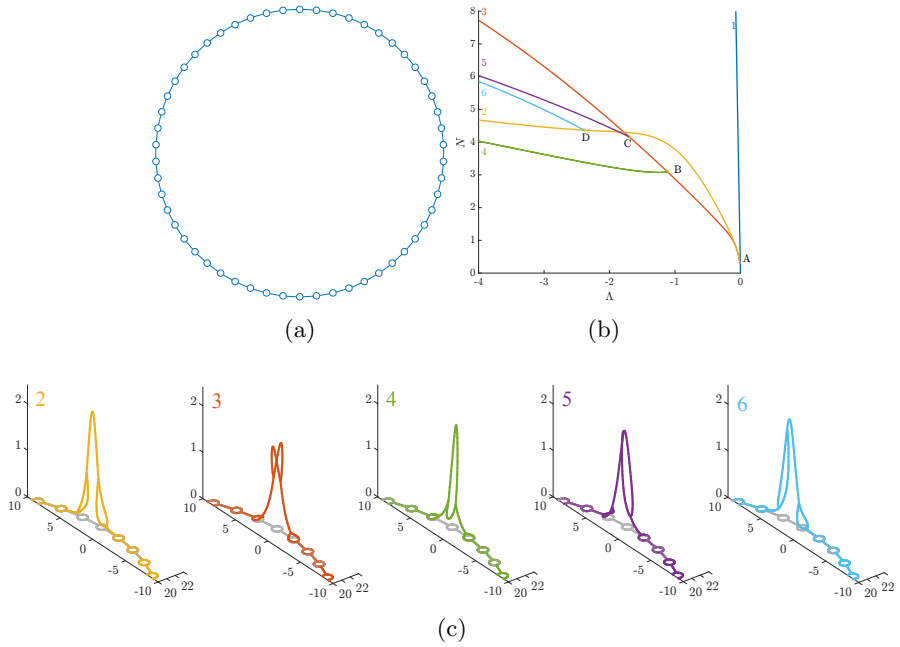


FIG. 2.7. (a) *Layout of the necklace graph, with* 54 *"strings" and* 54 *"pearls."* (b) *Partial bifurcation diagram plotting the solution's mass as a function of its frequency.* (c) *Solutions along the color-coded branches with frequency* $\Lambda \approx -4$.

lines of code to include in the published article, so these programs are understood most easily from the live script `continuationInstructions.mlx`. Figure 2.7(a) shows a so-called necklace graph, similar to an example in [14], consisting of 54 segments alternating between "strings" and "pearls." The cited paper allows the numerical domain to widen as the amplitude decreases, demonstrating the ground state scaling at small amplitude. However, because it does not employ continuation and branch-switching, it does not demonstrate the relationships between the branches.

Panel (b) of the figure shows a partial bifurcation diagram of standing wave solutions to cubic NLS, plotting $N = \|\Psi\|_2^2$ as a function of Λ. Branch 1 contains solutions with $\Psi(x, \Lambda)$ constant on Γ. At the point marked A, branches 2 and 3 bifurcate from the first branch. Solutions on branch 2 are symmetric around their maxima on the center of a string and those on branch 3 are symmetric around the centers of the pearls. At points B, C, and D, new branches arise due to symmetry-breaking (pitchfork) bifurcations. The figure shows five solutions with frequency $\Lambda \approx -4$. Solution 4 is the minimal mass solution at that frequency.

2.4. Time-stepping for evolution problems posed on a quantum graph. Our goal in this section is to adapt well-known time-dependent PDE solvers to work on quantum graphs using the framework described above for stationary problems. The first subsection contains two examples, while the second discusses the construction of a general solver for a certain class of such problems.

2.4.1. Elementary methods. We first adapt two standard methods to quantum graphs: the Crank–Nicholson method for the heat equation and the leapfrog method for the sine-Gordon equation. In the following, τ is the time step, $t_n = n\tau$

is the discretized times, and ψ_n represents the solution at time n discretized in time only and $\boldsymbol{\psi}_n$ the spatially discretized solution at time n.

Crank–Nicholson for the heat equation. The Crank–Nicholson method is a common second-order method for the heat equation $\frac{\partial \psi}{\partial t} = \triangle \psi$. Discretizing only in time, the update at time t_{n+1} is found by solving the equation

$$\frac{\psi_{n+1} - \psi_n}{\tau} = \frac{\triangle \psi_n + \triangle \psi_{n+1}}{2}.$$

In QGLAB, this is discretized and evaluated on the interior grid to give

$$\left(\mathbf{P}_{\text{int}} - \frac{\tau}{2}\mathbf{L}_{\text{int}}\right)\boldsymbol{\psi}_{n+1} = \left(\mathbf{P}_{\text{int}} + \frac{\tau}{2}\mathbf{L}_{\text{int}}\right)\boldsymbol{\psi}_n.$$

Combining this with homogeneous vertex conditions yields

$$\mathbf{L}_-\boldsymbol{\psi}_{n+1} \equiv \left(\mathbf{P}_{\text{VC}} - \frac{\tau}{2}\mathbf{L}_0\right)\boldsymbol{\psi}_{n+1} = \left(\mathbf{P}_0 + \frac{\tau}{2}\mathbf{L}_0\right)\boldsymbol{\psi}_n = \mathbf{L}_+\boldsymbol{\psi}_n. \tag{2.26}$$

The method iterates the MATLAB code `y = Lminus (Lplus*y)`. An example in section SM1.2.1 in the supplement computes a solution to the heat equation on the dumbbell graph and demonstrates convergence.

Leapfrog for nonlinear Klein–Gordon equations. The following example, contained in the live script `sineGordonOnTetra.mlx`, solves the sine-Gordon equation on the tetrahedron quantum graph, considered previously in [27], which consists of a wave equation with a sinusoidal nonlinearity,

$$\psi_{tt} - \triangle \psi + \sin \psi = 0. \tag{2.27}$$

Discretizing in time only and applying second-order centered differences in time gives

$$\frac{\psi_{n+1} - 2\psi_n + \psi_{n-1}}{\tau^2} = \triangle \psi_n - \sin \psi_n. \tag{2.28}$$

Applying the discretization in space and solving gives

$$\mathbf{P}_{\text{int}}\boldsymbol{\psi}_{n+1} = \mathbf{P}_{\text{int}}\left(2\boldsymbol{\psi}_n - \boldsymbol{\psi}_{n-1} - \tau^2 \sin \boldsymbol{\psi}_n\right) + \tau^2 \mathbf{L}_{\text{int}}\boldsymbol{\psi}_n$$

and enforcing the vertex conditions gives

$$\mathbf{P}_{\text{VC}}\boldsymbol{\psi}_{n+1} = \mathbf{P}_0\left(2\boldsymbol{\psi}_n - \boldsymbol{\psi}_{n-1} - \tau^2 \sin \boldsymbol{\psi}_n\right) + \tau^2 \mathbf{L}_0\boldsymbol{\psi}_n. \tag{2.29}$$

The iteration requires an approximation $\boldsymbol{\psi}_1 \in \mathbb{F}_0^{\text{ext}}$ at time t_1, which may be found from the initial conditions $\boldsymbol{\psi}|_{t=0} = \boldsymbol{\psi}_0$ and $\frac{\partial}{\partial t}\boldsymbol{\psi}\big|_{t=0} = \boldsymbol{\phi}_0$ using $O(\tau^2)$ approximation

$$\mathbf{P}_{\text{VC}}\boldsymbol{\psi}_1 = \mathbf{P}_0\left(\boldsymbol{\psi}_0 + \tau\boldsymbol{\phi}_0 - \frac{\tau^2}{2}\sin \boldsymbol{\psi}_0\right) + \frac{\tau^2}{2}\mathbf{L}_0\boldsymbol{\psi}_0.$$

The time-stepper in (2.29) is implemented by the line

```
u2 = PVC \ (P0*(2* u1 - u0 - tau^2*sin(u1)) + tau^2*L0*u1).
```

In section SM1.2.2 in the supplement, we compute an example from section 4.1 of [27] in which solitary waves collide with vertices on a tetrahedron metric graph and demonstrate convergence.

2.4.2. A general-purpose higher-order time-stepper. We now construct a general-purpose solver for differential equations of the form

$$\frac{\partial \psi}{\partial t} = \mu \triangle \psi + f(\psi), \tag{2.30}$$

posed on the quantum graph subject to any homogeneous vertex conditions implemented in QGLAB and such that $f(\psi)$ contains any terms involving a potential or nonlinearity. Depending on the constant μ, which could be real, imaginary, or complex, this formulation includes heat, Schrödinger, Ginzburg–Landau, and scalar reaction-diffusion equations. The vertex condition constraints make the straightforward application of standard methods somewhat difficult. Here, we construct a method that overcomes these problems.

The main issues in constructing a time-stepping algorithm for an evolutionary PDE defined on a quantum graph using the spatial discretization described in section 2.1 can be illustrated using Euler methods. These ideas then extend straightforwardly to Runge–Kutta algorithms. The forward Euler method is

$$\psi_{n+1} = \psi_n + \tau \left(\mu \triangle \psi_n + f(\psi_n) \right),$$

subject to vertex conditions applied to ψ_{n+1}. After we discretize in space and enforce the vertex conditions, this yields a time-stepper

$$\mathbf{P}_{\mathrm{VC}} \boldsymbol{\psi}_{n+1} = \mathbf{P}_0 \cdot \left(\boldsymbol{\psi}_n + \tau \mathbf{f}\left(\boldsymbol{\psi}_n \right) \right) + \tau \mu \mathbf{L}_0 \boldsymbol{\psi}_n. \tag{2.31}$$

The matrix $\mathbf{P}_{\mathrm{VC}}$ on the left makes the method implicit, but the implicitness is linear, requiring no Newton iterations. However, the stiff Laplacian term on the right is evaluated explicitly, imposing a step-size restriction, so the method is impractical.

Similarly, the backward Euler method is

$$\psi_{n+1} = \psi_n + \tau \left(\mu \triangle \psi_{n+1} + f(\psi_{n+1}) \right),$$

subject to vertex conditions on ψ_{n+1}. After we discretize in space and enforce the vertex conditions, this yields a time-stepper

$$\left(\mathbf{P}_{\mathrm{VC}} - \tau \mu \mathbf{L}_{\mathrm{VC}} \right) \boldsymbol{\psi}_{n+1} - \tau \mathbf{P}_0 \mathbf{f}(\boldsymbol{\psi}_{n+1}) = \mathbf{P}_0 \boldsymbol{\psi}_n. \tag{2.32}$$

The stiff Laplacian term is handled implicitly and imposes no time-step restrictions, but the implicit nonlinear term requires Newton iterations at each step and slows down the method.

To resolve this difficulty, we may treat the stiff term involving the Laplacian implicitly and the nonstiff term involving the nonlinearity explicitly. This idea was introduced for Runge–Kutta methods, which include the Euler method, by Ascher, Ruuth, and Spiteri [4]. There exist several such implicit-explicit (IMEX) Euler methods, including one they call forward-backward Euler $(1, 1, 1)$:

$$\psi_{n+1} = \psi_n + \tau \left(\mu \triangle \psi_{n+1} + f(\psi_n) \right)$$

subject to vertex conditions on ψ_{n+1}. After we discretize in space and enforce the vertex conditions, this yields a time-stepper

$$\left(\mathbf{P}_{\mathrm{VC}} - \tau \mu \mathbf{L}_{\mathrm{VC}} \right) \boldsymbol{\psi}_{n+1} = \mathbf{P}_0 \cdot \left(\boldsymbol{\psi}_n + \tau \mathbf{f}(\boldsymbol{\psi}_n) \right). \tag{2.33}$$

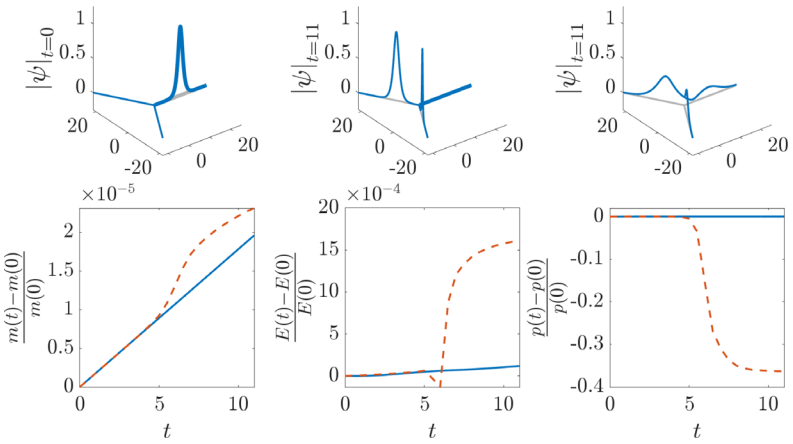


FIG. 2.8. *Collision of an NLS soliton with the vertex of a star graph. Top: (Left) Initial time. (Center) Final time, balanced graph. (Right) Final time, unbalanced graph. Bottom: Relative change in mass (left), energy (center), and momentum (right) with the balanced graph in solid blue and the unbalanced graph in dashed red.*

This method combines the best aspects of the forward and backward Euler methods. Moving the operator $\tau\mu\mathbf{L}_{\mathrm{VC}}$ to the left-hand side resolves the stiffness issue without requiring τ to be small. Keeping the nonlinear term on the right-hand side eliminates the need to solve a nonlinear equation on each step. However, the method is only first order in time, requiring small time steps for accuracy.

Reference [4] applies similar ideas to derive IMEX Runge–Kutta methods, which at each stage handle the stiff part of the evolution equation implicitly and the nonstiff part explicitly. The implicit terms are *strongly diagonal,* so each substage of a time step can be found in terms of the previous substages. QGLAB comes with a four-stage third-order Runge–Kutta method `qgdeSDIRK443`, based on the method denoted $(4, 4, 3)$ in [4].

Figure 2.8 shows the collision of a soliton with a vertex on a star graph with both so-called balanced and unbalanced Kirchhoff conditions, as considered in [39] and explained further in section 4.3. The soliton passes through the balanced vertex but is largely reflected by the unbalanced vertex. NLS dynamics conserves the mass (squared L^2 norm (1.5)) and energy (1.11); for certain initial conditions, the dynamics on the balanced graph also conserve momentum.

3. Understanding the MATLAB implementation.

3.1. The MATLAB digraph class. MATLAB provides tools for computations on undirected and directed graphs. The main component of QGLAB is a class `quantumGraph`, which builds upon the MATLAB `digraph` class used for defining directed-graph objects. The graph in Figure 1.1 is constructed using

```
source=[1 1 1 2 2]; target=[1 1 2 2 3];
G = digraph(source, target);
plot(G) % Actual figure created using quantumGraph plot function
```

The vectors `source` and `target` specify, respectively, the initial and final vertices of the four edges. The digraph object `G` contains two fields, `G.Edges` and `G.Nodes`, each in the form of a table—a MATLAB array type that holds column-oriented data, each

column stored as a variable. The methods `G.numnodes` and `G.numedges` return the number of vertices (nodes) and edges, respectively. The array of edges contains one variable `G.Edges.EndNodes`, an $|\mathcal{E}|\times 2$ array whose two columns contain, respectively, the indices of the source vertices and the target vertices. The nodes table is initially empty. We add fields to the two tables to create the quantum graph class.

3.2. Understanding the quantum graph class and initializing a quantum graph object. The graph and matrix shown in Figure 2.2 were generated with the code

```
source=[1 2]; target=[2 2]; L=[4 2*pi]; nx=[4 8];
G=quantumGraph(source,target,L,'nxVec',nx)
```

The last line initializes a `quantumGraph` object. The three required arguments `source`, `target`, and `L` must be entered in that order. The last, `nx`, is an optional argument. If `nx` is a vector of length `G.numedges`, it defines the number of interior points on each edge. If it is a scalar, the constructor will assign `nx` points per unit length to each edge, rounding if necessary.

The constructor takes several optional arguments, which will be discussed below. Some have default values if not specified in the function call. Optional arguments are listed in the function call using a key/value syntax after required arguments. In older releases of MATLAB, this is entered as `G=quantumGraph(source,target,L,'key1',value1,'key2',value2)`, while more recent releases allow the more compact syntax `G=quantumGraph(source,target,L,key1=value1,key2=value2)`. Complete instructions, including optional arguments, are presented in section SM2 of the supplement.

The above commands return the following in the MATLAB command window:

```
G =  quantumGraph with properties:

            discretization: 'Uniform'
       wideLaplacianMatrix: [12x16 double]
       interpolationMatrix: [12x16 double]
          discreteVCMatrix: [4x16 double]
    nonhomogeneousVCMatrix: [16x2 double]
          derivativeMatrix: [16x16 double]
                 potential: []
```

The most important property of this `quantumGraph` object is `qg`, which specifies the quantum graph itself and its discretization. It is not visible in the above listing because it is a *private* property of the object. The user cannot directly access it, but class methods may act on it. We will discuss it last. The remaining properties are publicly viewable but can only be set by class methods. They are as follows:

- `discretization` may take three values, `'Uniform'` (default), `'Chebyshev'`, or `'None'`. With the `'None'` property, only secular determinant computations function.
- `wideLaplacianMatrix` The rectangular Laplacian matrix $\mathbf{L}_{\rm int}$ defined in (2.12) for either the uniform or Chebyshev discretization.
- `interpolationMatrix` The interpolation or resampling matrix $\mathbf{P}_{\rm int}$ as defined in (2.13) for either the uniform or the Chebyshev discretization.
- `discreteVCMatrix` The matrix $\mathbf{M}_{\rm VC}$ defining the discretization of the vertex conditions defined in (2.12) and (2.13) for either discretization.

- `nonhomogeneousVCMatrix` The matrix $\mathbf{M}_{\mathrm{NH}}$ defined in (2.14) which maps nonhomogeneous terms in the vertex condition to the appropriate row.
- `derivativeMatrix` The square matrix used to calculate the first derivative on each edge. It is used to compute time-dependent solutions' energy and momentum functionals and to plot solution branches in continuation problems.
- `potential` The optional potential $V(x)$.

The property `qg` is a MATLAB directed-graph object consisting of a `Nodes` table and an `Edges` table with added fields needed to define a quantum graph. Because `qg` is a private property, viewing these tables using the syntax `G.qg.Nodes` and `G.qg.Edges` is disabled. Instead, `Nodes` and `Edges` `quantumGraph` methods have been written that return each of these tables, so we may view the tables using the syntax `G.Nodes` and `G.Edges`, as in this code listing. In addition, a method exists that returns each default table column; for example, the Robin coefficients can be returned by `G.robinCoeff`. We examine the node data, which has two fields:

```
> >  disp(G.Nodes)
    robinCoeff     y
    ----------    ---
        0          NaN
        0          NaN
```

The fields are as follows:

- `robinCoeff` The Robin coefficients α_n used to define the vertex condition (1.3). To implement a Dirichlet vertex condition (1.4), use a not-a-number (`NaN`). Default value 0 for Neumann–Kirchhoff conditions.
- `y` The values of ψ at the vertices, set to `NaN` on initialization. Used only for plotting.

We then examine the edges table, which has seven required fields:

```
> >  disp(G.Edges)
    EndNodes Weight   L    nx        x              y      Field7
    -------- ------  ----  --  ------------  ------------   ----
     1  2       1      4    4  { 6x1 double}  { 6x1 double}    1
     2  2       1    6.28   8  {10x1 double}  {10x1 double}  0.79
```

The fields are as follows:

- `EndNodes` This $|\mathcal{E}|\times 2$ array contains the indices of the initial and final vertices of the edges, i.e., the content of the input variables `source` and `target`.
- `Weight` The weight w_j in vertex condition (1.3). Defaults to one if unset.
- `L` The array of edge lengths.
- `nx` The number of discretization points on the interior of each edge.
- `x` The (`nx` + 2) discretization points on each edge, including ghost points for the uniform discretization and vertices in the Chebyshev discretization.
- `y` The value of ψ at the discretization points, initially set to `NaN`.
- `Field7` This field, named `integrationWeight`, is the spatial discretization `dx` for uniform discretizations and the Curtis–Clenshaw weights used to compute integrals with the Chebyshev discretization.

Plotting coordinates are defined in a separate program, e.g., Figure 2.2(a) was created using the program `lassoPlotCoords.m`. The coordinates are assigned to the graph `G` with the command `G.addPlotCoords(@lassoPlotCoords)` after `G` has been

created. Alternately, it can be included in the constructor by setting the optional argument `PlotCoordinateFcn` to the value `@lassoPlotCoords`.

The plot coordinates are stored in fields `x1` and `x2` in both the `Edges` and `Nodes` tables. The MATLAB `plot` command is overloaded so `G.plot` plots the `y` coordinate over a skeleton of the graph in the `x1` and `x2` coordinates. Some graphs, such as those formed from the edges and vertices of a platonic solid, are best depicted in three space variables, so the user may define a third plot coordinate `x3`. If `x3` exists, the graph is plotted in three dimensions with the `y` coordinate represented by a color scale.

QGLAB includes templates for various commonly studied graphs and a template syntax that allows the quick creation of such graphs, such as the lasso and tetrahedron templates used in section 3.3.1. These have default parameters that can be overridden. A gallery of graph templates is included in the documentation.

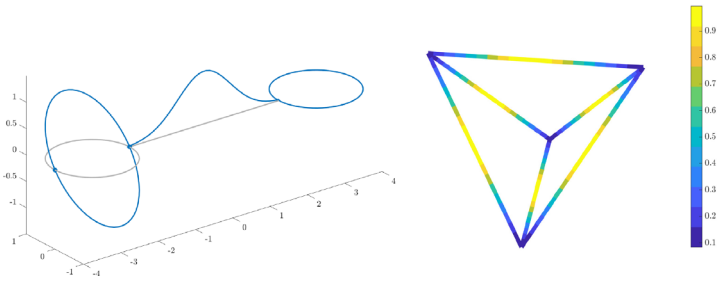
3.3. Basic operations. A MATLAB live script is a rich document that includes runnable code and formatted text, entered with a simple word processor–like interface, and which integrates outputs including text and graphics, which can be exported to formats including PDF, LaTeX, and HTML. QGLAB includes many examples created as live scripts and exported to HTML. Basic operations are described in the file `documentation/quantumGraphRoutines.mlx`.

3.3.1. Function evaluation/plotting. The command `applyFunctionToEdge` evaluates a function specified by a function handle, anonymous function, or constant value and assigns its value to edge e_j. The command `applyFunctionsToAllEdges` applies a cell array of functions to all the edges; for example, the following commands define and plot a dumbbell quantum graph with the default parameters, plotted in Figure 3.1(a):

```
G=quantumGraphFromTemplate('dumbbell');
G.applyFunctionsToAllEdges(@sin,@(x)exp(-(x-2).2),0);
G.plot
```

QGLAB also provides some three-dimensional templates, for which the `y` values are plotted using a color scale, as in Figure 3.1(b), where we plot Gaussians on all edges of a regular tetrahedron using the commands

```
G=solidTemplates('tetrahedron');
f=@(x)(exp((-10*(x-.5).2)));
G.applyFunctionsToAllEdges(f,f,f,f,f,f);
G.plot;
```



(a) (b)

FIG. 3.1. (a) *A function defined on the edges of a dumbbell graph.* (b) *A function defined on the edges of a tetrahedral graph.*

Finally, on some graphs with many edges, plots in three dimensions become a confusing tangle of curves, and it is more illuminating to visualize them as a color scale, using an overloaded `pcolor` command, demonstrated in section SM2.

3.3.2. Getting data on and off the graph. The discretized numerical problems, including the Poisson problem (2.15) and the eigenvalue problem (2.24), are posed in terms of unknown column vectors. By contrast, this data is stored edge by edge in the `quantumGraph` object in `G.qg.Edges.y`. The command `graph2column` creates such a column vector from the data in graph `G`, while the command `column2graph` loads the data from a column vector onto the edges of the graph including the vertices (Chebyshev) or ghost points (uniform). Under the uniform discretization, the command also interpolates the data to the vertices. The `column2graph` command is also called by the command `G.plot(y)` to plot the contents of the vector `y` over the graph's skeleton.

3.3.3. Other overloaded functions. Many methods from the MATLAB directed-graph class have been overloaded so that, for example, a call to `G.numnodes` returns the number of nodes and `G.numedges` the number of edges. An overloaded `spy` command (along with additional formatting) was used to visualize the Laplacian matrix in Figures 2.2 and 2.4. Overloaded versions of the `norm` and `dot` commands are used frequently throughout the package. Section SM2 of the supplement lists these functions.

4. Extended examples. Here, we give some brief examples of problem setup and accuracy. The supplement contains some additional details and other examples. A MATLAB live script for each, featuring plots and a convergence study, is included in the QGLAB GitHub repository and indexed in the **readme** file [34].

4.1. The Poisson problem. We construct a discretization (2.15) of the Poisson problem (2.10) using the graph in Figure 1.1 with edges of lengths $(\pi, 2\pi, 1, 2\pi, 2)$. The vertex conditions at $\mathtt{v}_1$ and $\mathtt{v}_2$ are Kirchhoff–Robin with $\alpha_1 = \alpha_2 = 1$ and at $\mathtt{v}_3$ the vertex condition is Dirichlet. The weight vector is $\mathbf{w} = (1, 1, 2, 1, 1)$. The potential vanishes except on edge $\mathtt{e}_1$ where $V_1 = 2\cos 2x$. The nonhomogeneous terms are

$$f = \left(-\sin 3x, 2\cos 2x, -4; -\sin x, \operatorname{sech} x - 2\operatorname{sech}^3 x\right) \quad \text{and} \quad \psi = (8, 3, \operatorname{sech} 2).$$

The exact solution is

$$\psi = \left(\sin x, \sin^2 x, 3x - 2x^2, 1 + \sin x, \operatorname{sech} x\right).$$

A minimal code to solve this problem is

```
1 s=[1 1 1 2 2]; t=[1 1 2 2 3]; L=[pi 2*pi 1 2*pi 2];
2 rc = [1 1 nan]; w = [1 1 2 1 1]; V = {@(x)(2*cos(2*x)),0,0,0,0};
3 G=quantumGraph(s,t,L,'RobinCoeff',rc,'Weight',w,'Potential',V);
4 f = G.applyFunctionsToAllEdges({@(x)-sin(3*x);@(x)2*cos(2*x);...
5     -4;@(x)-sin(x);@(x)sech(x)-2*sech(x).^3});
6 phi = [8;3;sech(2)];
7 psi = G.solvePoisson('edgeData',f,'nodeData',phi);
```

Lines 1 and 2 define the quantum graph, the parameters that define the vertex conditions, and the potential. The variables `s`, `t`, and `L` define the source vertices, target vertices, and lengths of the five edges. The first two elements of `rc` define the Robin coefficients at vertices $\mathtt{v}_{1,2}$, and the third component is `nan` (not-a-number), indicating the Dirichlet condition. The weight vector and the potential are given by `w` and `V`.

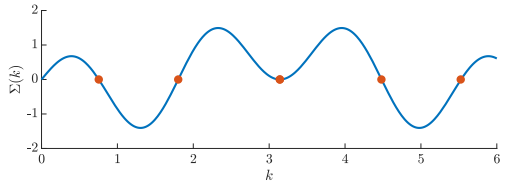


FIG. 4.1. *The secular determinant $\Sigma(k)$, of the Y-shaped graph discussed in the text, along with the computed values $k_j = \sqrt{-\lambda_j}$, which sit right on top of the zeros.*

Line 3 constructs the quantum graph from this data. It used the default centered-difference discretization and the default value `nX = 20`, so it uses $h = \frac{1}{20}$ or a slightly smaller value on each edge to achieve an integer number of subintervals per edge. Lines 4–6 define the nonhomogeneous terms, and line 7 solves the discretized problem.

The maximum pointwise error of this solution is 1.02×10^{-3}. Doubling the resolution by inserting the arguments `'nX',40` at line 3 reduces the error to 2.56×10^{-4}, a factor of 4.01. This is consistent with the expected second-order convergence. Adding the arguments `'Discretization','Chebyshev'` at line 3 switches to Chebyshev discretization. With 16 points per edge, this gives a maximum error of 1.80×10^{-7}; with 32, the error is 1.56×10^{-12}. This is consistent with spectral convergence.

4.2. Eigenproblems. The example plotted in Figure 2.5 is computed in the live script `starEigenfunctionsDemo.mlx`. The Y-shaped graph has edges of lengths $\{\frac{3}{2}, 1, 1\}$. Its Kirchhoff conditions at both ends of the longer edge $\mathtt{e}_1$ and Dirichlet conditions at the remaining vertices are defined by setting the first two elements of the `robinCoeff` vector to zero the remaining two to `nan`:

```
1 G = quantumGraphFromTemplate('star','LVec',LVec=[1.5 1 1],...
2                              'robinCoeff',[0 0 nan nan],'nX',40);
3 [V,lambda]=G.eigs(4); % Compute 4 eigenvalues
4 for k=1:4; G.plot(V(:,k));end;
```

Running `G.secularDet` returns the following secular determinant plotted in Figure 4.1,

$$\Sigma(k) = \frac{4}{3} \sin \frac{k}{2} (\cos k + 1) \left(6 \cos^2 k - 3 \cos k - 1\right).$$

The $\Sigma(k)$ plot shows that the eigenvalue $\lambda = -\pi^2 \approx -0.9865$ has multiplicity two. Generally, a numerical eigenvalue solver will return two closely spaced eigenvalues rather than a double eigenvalue. The graph Γ is symmetric under the interchange of the edges $\mathtt{e}_2$ and $\mathtt{e}_3$. The multiplicity-one eigenfunctions respect this symmetry, but the multiplicity-two eigenfunctions returned by `eigs` do not. The script takes appropriate linear combinations of the two computed eigenvectors to produce odd and even eigenvectors with respect to this symmetry; see panels 3 and 4 of Figure 2.5. Section SM1.1.1 of the supplement discusses the convergence, which is standard.

4.3. Time-dependent problems. The NLS equation (1.9) is the prototype of a dispersive nonlinear PDE and many studies have considered its evolution on quantum graphs. Kairzhan, Pelinovsky, and Goodman consider the cubic NLS equation posed on a "star graph" consisting of three half-lines joined at a single vertex [39]. The evolution conserves mass, i.e., the squared L^2 norm (1.5), and the energy (1.11), but in general, it does not conserve momentum. If, however, the parameters w_m are appropriately chosen in the weighted Kirchhoff–Robin vertex condition (1.3) to form

a so-called balanced star graph, and the initial condition is chosen to lie in a particular invariant subspace, then the dynamics on this graph do conserve momentum.

We simulate the collision of a soliton propagating on edge $\mathbf{e}_1$ toward the vertex in Figure 2.8 using the IMEX Runge–Kutta solver `qgdeSDIRK443`. The first simulation is computed on a "balanced" graph whose vertex condition is defined by the weight vector $\mathbf{w} = (2, 1, 1)$ in the first line of the following script.

```
G =quantumGraphFromTemplate('star','LVec',30,'weight',[2 1 1]);
soliton =@(x,t,v,x0)exp(1i*(-v*x/2-(1-v^2/4)*t)).*sech(x-x0-v*t);
init1 = @(x)soliton(x,0,-2,15); init2 =@(x)soliton(x,0,2,-15);
u0 = G.applyFunctionsToAllEdges({init1,init2,init2});
mu = -1i; F =@(z) -2i * z.^2.* conj(z);
tFinal = 11; dt = 0.01; tPrint =0.5; nSkip = tPrint/dt;
[t,u] = G.qgdeSDIRK443(mu,F,tFinal,u0,dt,'nSkip',nSkip);
```

The soliton splits into two, each new soliton propagating along the edge with its original amplitude and velocity. At this discretization, the mass, energy, and momentum are all conserved to 5, 4, and 4 digits, respectively.

We next set $\mathbf{w} = (1, 1, 1)$. Now, a significant fraction of the soliton is reflected and propagates backward along the incoming edge. Mass and energy conservation are slightly worse, but momentum changes by $O(1)$. The standard test of computing the numerical solution with time steps τ, $\tau/2$, and $\tau/4$ shows that the difference between subsequent solutions decreases about eight times, indicating third-order convergence.

5. Conclusions. QGLAB is a robust and versatile MATLAB package for computing solutions to the spectral accuracy of linear and nonlinear problems on quantum graphs. It allows users to build graph models quickly, analyze their spectrum, compute nonlinear bifurcations, and solve evolution equations. The algorithms are implemented at a high level, hiding most implementation details and allowing the user to focus on the mathematical problem, not the numerical and algorithmic details.

Linear and nonlinear PDEs on quantum graphs remain a vibrant area of analysis in spectral geometry in which the interaction of geometry, topology, and symmetry gives rise to diverse mathematical questions [2, 10, 31] with many open problems left to explore. Many previously studied problems on combinatorial graphs have analogies on metric graphs that remain open and where the spectrum of behaviors is likely to be much richer. For example, the spectral optimization of combinatorial graphs has been studied in [52], and others have examined how the symmetries of discrete Laplacians can lead to interesting spectral features such as Dirac points and flat bands [42, 46]. The study of time-dependent evolution equations on quantum graphs remains in its infancy [27, 39]. QGLAB is an ideal tool for exploring these problems.

Reproducibility of computational results. This paper has been awarded the "SIAM Reproducibility Badge: Code and data available" as a recognition that the authors have followed reproducibility principles valued by SISC and the scientific computing community. Code and data that allow readers to reproduce the results in this paper are available at https://github.com/manroygood/Quantum-Graphs/ and in the supplementary materials (Quantum-Graphs-master.zip [local/web 13.5MB]).

Acknowledgments. The authors thank Greg Berkolaiko, Dmitry Pelinovsky, David Shirokoff, and Nick Trefethen for helpful conversations. RG credits a semester-long funded visit to the IMA at the University of Minnesota in 2016 for providing the time and inspiration to begin work on quantum graphs.

REFERENCES

[1] R. Adami, E. Serra, and P. Tilli, *Nonlinear dynamics on branched structures and networks*, Riv. Math. Univ. Parma, 8 (2017), pp. 109–159, https://www.rivmat.unipr.it/vols/2017-8-1/adami-et-al.html.

[2] L. Alon, R. Band, and G. Berkolaiko, *Nodal statistics on quantum graphs*, Comm. Math. Phys., 362 (2018), pp. 909–948, https://doi.org/10.1007/s00220-018-3111-2.

[3] M. Arioli and M. Benzi, *A finite element method for quantum graphs*, IMA J. Numer. Anal., 38 (2018), pp. 1119–1163, https://doi.org/10.1093/imanum/drx029.

[4] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations*, Appl. Numer. Math., 25 (1997), pp. 151–167, https://doi.org/10.1016/S0168-9274(97)00056-1.

[5] J. L. Aurentz and L. N. Trefethen, *Block operators and spectral discretizations*, SIAM Rev., 59 (2017), pp. 423–446, https://doi.org/10.1137/16M1065975.

[6] R. Band, G. Berkolaiko, H. Raz, and U. Smilansky, *The number of nodal domains on quantum graphs as a stability index of graph partitions*, Comm. Math. Phys., 311 (2012), pp. 815–838, https://doi.org/10.1007/s00220-011-1384-9.

[7] T. Beck, I. Bors, G. Conte, G. Cox, and J. L. Marzuola, *Limiting eigenfunctions of Sturm–Liouville operators subject to a spectral flow*, Ann. Math. Qué., 49 (2020), pp. 249–269, https://doi.org/10.1007/s40316-020-00142-6.

[8] S. Becker, F. Gregorio, and D. Mugnolo, *Schrödinger and polyharmonic operators on infinite graphs: Parabolic well-posedness and p-independence of spectra*, J. Math. Anal. Appl., 495 (2021), 124748, https://doi.org/10.1016/j.jmaa.2020.124748.

[9] G. Berkolaiko, *An elementary introduction to quantum graphs*, in Geometric and Computational Spectral Theory, Contemp. Math. 700, AMS, Providence, RI, 2017, pp. 41–72.

[10] G. Berkolaiko, J. Kennedy, P. Kurasov, and D. Mugnolo, *Surgery principles for the spectral analysis of quantum graphs*, Trans. Amer. Math. Soc., 372 (2019), pp. 5153–5197, https://doi.org/10.1090/tran/7864.

[11] G. Berkolaiko and P. Kuchment, *Introduction to Quantum Graphs*, Math. Surveys .Monographs 186, AMS, Providence, RI, 2013.

[12] G. Berkolaiko, J. L. Marzuola, and D. E. Pelinovsky, *Edge-localized states on quantum graphs in the limit of large mass*, Ann. Henri Poincaré C, 38 (2020), pp. 1295–1335, https://doi.org/10.1016/j.anihpc.2020.11.003.

[13] J.-P. Berrut and L. N. Trefethen, *Barycentric Lagrange interpolation*, SIAM Rev., 46 (2004), pp. 501–517, https://doi.org/10.1137/S0036144502417715.

[14] C. Besse, R. Duboscq, and S. Le Coz, *Numerical simulations on nonlinear quantum graphs with the GraFiDi library*, SMAI J. Comput. Math., 8 (2021), pp. 1–47, https://doi.org/10.5802/smai-jcm.78.

[15] W. Borrelli, R. Carlone, and L. Tentarelli, *Nonlinear Dirac equation on graphs with localized nonlinearities: Bound states and nonrelativistic limit*, SIAM J. Math. Anal., 51 (2019), pp. 1046–1081, https://doi.org/10.1137/18M1211714.

[16] W. Borrelli, R. Carlone, and L. Tentarelli, *An overview on the standing waves of nonlinear Schrödinger and Dirac equations on metric graphs with localized nonlinearity*, Symmetry, 11 (2019), 169, https://doi.org/10.3390/sym11020169.

[17] D. Borthwick, E. M. Harrell II, and K. Jones, *The heat kernel on the diagonal for a compact metric graph*, Ann. Henri Poincaré, 24 (2023), pp. 1661–1680, https://doi.org/10.1007/s00023-022-01248-z.

[18] L. Böttcher and M. A. Porter, *Dynamical Processes on Metric Networks*, https://arxiv.org/abs/2401.00735, 2024.

[19] M. Brio, J.-G. Caputo, and H. Kravitz, *Spectral solutions of PDEs on networks*, Appl. Numer. Math., 172 (2022), pp. 99–117, https://doi.org/10.1016/j.apnum.2021.09.021.

[20] C. Cacciapuoti, S. Dovetta, and E. Serra, *Variational and stability properties of constant solutions to the NLS equation on compact metric graphs*, Milan J. Math., 86 (2018), pp. 305–327, https://doi.org/10.1007/s00032-018-0288-y.

[21] G. Conte, *Numerical Analysis of Linear and Nonlinear Schrödinger Equations on Quantum Graphs*, Ph.D. thesis, University of North Carolina, 2022.

[22] C. De Coster, S. Dovetta, D. Galant, and E. Serra, *On the notion of ground state for nonlinear Schrödinger equations on metric graphs*, Calc. Var. Partial Differential Equations, 62 (2023), 159, https://doi.org/10.1007/s00526-023-02497-4.

[23] S. Dovetta and L. Tentarelli, *Ground states of the L^2-critical NLS equation with localized nonlinearity on a tadpole graph*, in Discrete and Continuous Models in the Theory of Networks, Springer, New York, 2020, pp. 113–125.

[24] T. A. Driscoll and N. Hale, *Rectangular spectral collocation*, IMA J. Numer. Anal., 36 (2016), pp. 108–132, https://doi.org/10.1093/imanum/dru062.
[25] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun Guide*, Pafnuty Publications, 2014.
[26] Y. Du, B. Lou, R. Peng, and M. Zhou, *The Fisher-KPP equation over simple graphs: Varied persistence states in river networks*, J. Math. Biol., 80 (2020), pp. 1559–1616, https://doi.org/10.1007/s00285-020-01474-1.
[27] D. Dutykh and J.-G. Caputo, *Wave dynamics on networks: Method and application to the sine-Gordon equation*, Appl. Numer. Math., 131 (2018), pp. 54–71, https://doi.org/10.1016/j.apnum.2018.03.010.
[28] L. Edsberg, *Introduction to Computation and Modeling for Differential Equations*, Wiley, New York, 2016.
[29] P. Exner and O. Post, *Convergence of spectra of graph-like thin manifolds*, J. Geom. Phys., 54 (2005), pp. 77–115, https://doi.org/10.1016/j.geomphys.2004.08.003.
[30] P. Exner and O. Post, *A general approximation of quantum graph vertex couplings by scaled Schrödinger operators on thin branched manifolds*, Comm. Math. Phys., 322 (2013), pp. 207–227, https://doi.org/10.1007/s00220-013-1699-9.
[31] S. Gnutzmann and U. Smilansky, *Quantum graphs: Applications to quantum chaos and universal spectral statistics*, Adv. Phys., 55 (2006), pp. 527–625, https://doi.org/10.1080/00018730600908042.
[32] S. Gnutzmann and D. Waltner, *Stationary waves on nonlinear quantum graphs: General framework and canonical perturbation theory*, Phys. Rev. E, 93 (2016), 032204, https://doi.org/10.1103/PhysRevE.93.032204.
[33] N. Goloshchapova, *A nonlinear Klein–Gordon equation on a star graph*, Math. Nachr., 294 (2021), pp. 1742–1764, https://doi.org/10.1002/mana.201900526.
[34] R. Goodman and G. Conte, *Quantum-Graphs: v*1.2, 2024, https://github.com/manroygood/Quantum-Graphs/.
[35] R. H. Goodman, *NLS bifurcations on the bowtie combinatorial graph and the dumbbell metric graph*, Discrete Contin. Dyn. Syst. Ser. A, 39 (2019), pp. 2203–2232, https://doi.org/10.3934/dcds.2019093.
[36] W. J. F. Govaerts, *Numerical Methods for Bifurcations of Dynamical Equilibria*, SIAM, Philadelphia, 2000.
[37] D. Grieser, *Spectra of graph neighborhoods and scattering*, Proc. Lond. Math. Soc., 97 (2008), pp. 718–752, https://doi.org/10.1112/plms/pdn020.
[38] E. Harrell, *Spectral theory on combinatorial and quantum graphs*, in Théorie spectrale des graphes et des variétés, C. Anné and N. Torki-Hamza, eds., CIMPA, 2016.
[39] A. Kairzhan, D. E. Pelinovsky, and R. H. Goodman, *Drift of spectrally stable shifted states on star graphs*, SIAM J. Appl. Dyn. Syst., 18 (2019), pp. 1723–1755, https://doi.org/10.1137/19M1246146.
[40] H. B. Keller, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, Publ. Math. Res. 84, P. Rabinowitz, ed., Academic Press, New York, 1977.
[41] T. Kottos and U. Smilansky, *Quantum chaos on graphs*, Phys. Rev. Lett., 79 (1997), pp. 4794–4797, https://doi.org/10.1103/PhysRevLett.79.4794.
[42] L.-K. Lim, J.-N. Fuchs, F. Piéchon, and G. Montambaux, *Dirac points emerging from flat bands in Lieb-Kagome lattices*, Phys. Rev. B., 101 (2020), 045131, https://doi.org/10.1103/PhysRevB.101.045131.
[43] D. Maier, W. Reichel, and G. Schneider, *Breather solutions for a semilinear Klein-Gordon equation on a periodic metric graph*, J. Math. Anal. Appl., 528 (2023), 127520, https://doi.org/10.1016/j.jmaa.2023.127520.
[44] G. Malenova, *Spectra of Quantum Graphs*, Master's thesis, Czech Technical University in Prague, 2013.
[45] J. L. Marzuola and D. E. Pelinovsky, *Ground state on the dumbbell graph*, Appl. Math. Res. Express AMRX, 2016 (2016), pp. 98–145, https://doi.org/10.1093/amrx/abv011.
[46] L. Morales-Inostroza and R. A. Vicencio, *Simple method to construct flat-band lattices*, Phys. Rev. A, 94 (2016), 043831, https://doi.org/10.1103/PhysRevA.94.043831.
[47] D. Mugnolo, *Semigroup Methods for Evolution Equations on Networks*, Underst. Complex Syst., Springer, Cham, 2014.
[48] D. Mugnolo, D. Noja, and C. Seifert, *Airy-type evolution equations on star graphs*, Anal. PDE, 11 (2018), pp. 1625–1652, https://doi.org/10.2140/apde.2018.11.1625.
[49] A. H. Nayfeh and B. Balachandran, *Applied Nonlinear Dynamics: Analytical, Computational and Experimental Methods*, Wiley-VCH, Weinheim, 1995.

[50] D. Noja, *Nonlinear Schrödinger equation on graphs: Recent results and open problems*, Philos. Trans. Roy. Soc. A, 372 (2014), 20130002, https://doi.org/10.1098/rsta.2013.0002.

[51] D. Noja, D. Pelinovsky, and G. Shaikhova, *Bifurcations and stability of standing waves in the nonlinear Schrödinger equation on the tadpole graph*, Nonlinearity, 28 (2015), pp. 2343–2378, https://doi.org/10.1088/0951-7715/28/7/2343.

[52] B. Osting and J. Marzuola, *Spectrally optimized pointset configurations*, Constr. Approx., 46 (2017), pp. 1–35, https://doi.org/10.1007/s00365-017-9365-7.

[53] J. A. Pava and M. Cavalcante, *Linear instability criterion for the Korteweg–de Vries equation on metric star graphs*, Nonlinearity, 34 (2021), pp. 3373–3410, https://doi.org/10.1088/1361-6544/abea6b.

[54] K. K. Sabirov, D. Babajanov, D. U. Matrasulov, and P. G. Kevrekidis, *Dynamics of Dirac solitons in networks*, J. Phys. A, 51 (2018), 435203, https://doi.org/10.1088/1751-8121/aadfb0.

[55] E. Serra and L. Tentarelli, *On the lack of bound states for certain NLS equations on metric graphs*, Nonlinear Anal. Theory Methods Appl., 145 (2016), pp. 68–82, https://doi.org/10.1016/j.na.2016.07.008.

[56] Z. Sobirov, D. Babajanov, D. Matrasulov, K. Nakamura, and H. Uecker, *Sine-Gordon solitons in networks: Scattering and transmission at vertices*, Europhys. Lett., 115 (2016), 50002, https://doi.org/10.1209/0295-5075/115/50002.

[57] K. Xu and N. Hale, *Explicit construction of rectangular differentiation matrices*, IMA J. Numer. Anal., 36 (2016), pp. 618–632, https://doi.org/10.1093/imanum/drv013.