Politecnico di Milano

MSc in Computational Science and Engineering

Course in Advanced Programming for Scientific Computing
Course in Numerical Analysis for Partial Differential Equations

Finite Element Methods on Graphs

Authors:
Stefano Abbate (905615)
Andrea Di Primio (900358)

Supervisor:
Prof. Paolo Zunino

a.y. 2019/2020

POLITECNICO
MILANO 1863

Contents

Abstract

This report illustrates the work done for the joint final project in the courses of Advanced Programming for Scientific Computing (Prof. L. Formaggia) and Numerical Analysis for Partial Differential Equations (Prof. P. F. Antonietti), held at Politecnico di Milano during the academic year 2018/2019. The main aim of the project is to investigate how to solve elliptic partial differential equations on graphs from analytical, numerical and computational standpoints. The content of this document is structured as follows. Chapter 1 illustrates the necessary background in graph theory. Graph discretization and (linear) finite element methods are introduced in Chapter 2. A theoretical introduction to partial differential equations defined on graphs and their discrete versions is given in Chapter 3, with particular attention to elliptic equations and elliptic eigenvalue problems. Numerical experiments on different graph topologies are conducted and analysed in Chapter 4, through an original C++ implementation. In Chapter 5, a more in-depth theoretical analysis on tree graphs is given, with special interest to the spectrum of elliptic operators on trees. Further numerical experiments are also conducted in order to verify the so called Weyl asymptotic formula. Conclusions and remarks on further elaborations are given in Chapter 6. Finally, Chapter 7 is devoted to the implementation details of the C++ code used to deal with the algebraic versions of the differential equations described throughout the report.

Chapter 1

Elements of Graph Theory

1.1 Combinatorial graphs

Let us start by briefly recalling elementary concepts in graph theory. The topic presentation hereafter is by no means exhaustive and it is limited to the notions that will be employed in this work. A comprehensive introduction to graph theory can be found in [3]. The object "graph" is the mathematical abstraction of what can be intuitively represented as a set of points, called vertices, connected by a collection of curves, called edges (also branches or arcs). A simple example of graph is shown in FIGURE 1.1.

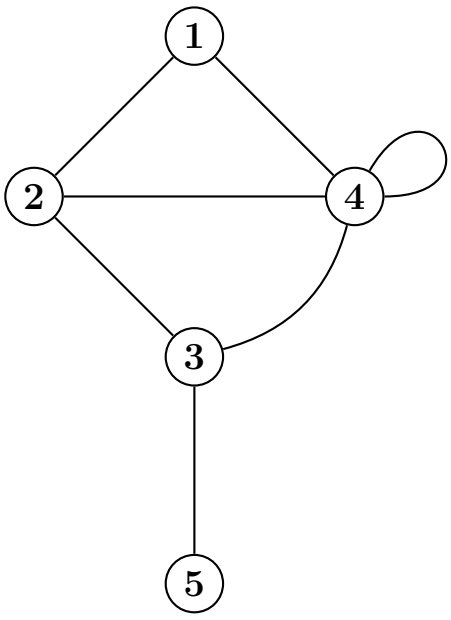


Figure 1.1: A graphical representation of a graph with 5 vertices and 7 edges.

A more rigorous formulation of the heuristic idea formulated above is contained in the following Definition.

Definition 1.1 (Combinatorial graph)**.** A combinatorial graph (from here on simply graph) Γ is a couple $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes a finite set of vertices $\{v_1, ..., v_N\}$ and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is a set of edges. Such graphs are said to be directed, and each edge (v_i, v_j) is understood to be oriented from v_i (called source) to v_j (called target). The graph is said to be undirected upon identification of the edges (v_i, v_j) and (v_j, v_i) for any valid value of the indices i and j.

Remark 1.1. DEFINITION 1.1 is not completely satisfactory: it exploits the intuitive idea of undirected graph, for instance. Nonetheless, for the purposes of this work, we find this approach preferable with respect to introducing further notions, like the one of 1-simplicial complex.

Remark 1.2. Notice that, according to DEFINITION 1.1, an edge is fully identified by a pair of vertices, namely by an element of $\mathcal{V} \times \mathcal{V}$. Thus, at the moment, edges have no kind of geometrical property. We also point out that $\mathcal{V}$ need not be a subset of $\mathbb{R}^n$.

By means of REMARK 1.2, the undirected graph in FIGURE 1.1 should be represented with the two sets

$$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}, \qquad \mathcal{E} = \{(v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4), (v_3, v_5), (v_4, v_4)\}$$

assuming vertex i in the picture is represented by the element v_i of $\mathcal{V}$.
The following Definitions address graphs with specific properties, to which we will mainly restrict our attention.

Definition 1.2 (Connected graph)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a graph. Γ is said to be connected if for every pair of vertices $v_i, v_j \in \mathcal{V}$ there exist an integer k, $k \geq 1$, and a sequence of vertices $v_{n_1}, ..., v_{n_k}$ in $\mathcal{V}$ such that $(v_i, v_{n_1}), (v_{n_1}, v_{n_2}), ..., (v_{n_{k-1}}, v_{n_k}), (v_{n_k}, v_j)$ belong to $\mathcal{E}$. We also say that there exists a path from v_i to v_j, $\forall i, j$.

In simpler words, a graph is connected if and only if every vertex can be, in fact, connected to every other vertex through existing edges. Notice that DEFINITION 1.2 can describe a property of both undirected and directed graphs. The example graph in FIGURE 1.1 is, as one can simply verify by hand, connected. If a graph is not connected, one can compute the number of its connected components as the number of equivalence classes of the quotient space $\mathcal{V}/\sim$ where $v \sim w$ iff there exists a path from v to w. A second class of graphs that will be of interest in the following is the one of sparse graphs.

Definition 1.3 (Sparse graph)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a graph. Γ is said to be sparse if, letting N be the number of its vertices and M the number of its edges, $M = O(N)$, where O stands for the big-O notation.

Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be an undirected and connected graph without loops (i.e. without edges of the type (v, v) with $v \in \mathcal{V}$). For the purposes of this work, directions on the edges of the graph are irrelevant. However, since dealing with directed graphs is convenient (as it will be clear later), we assign an arbitrary orientation to each edge through the construction of the so called incidence matrix of the graph. Following the notation of DEFINITION 1.3, N will denote the number of vertices and M the number of edges in the graph. We stress that N is assumed to be finite (and thus also M).

Definition 1.4 (Incidence matrix)**.** Given a directed graph $\Gamma = (\mathcal{V}, \mathcal{E})$, the incidence matrix of Γ is a matrix $\mathbf{E} \in \mathbb{R}^{N \times M}$ such that each column represents a single edge in the graph by having only two non-zero entries, 1 corresponding to the source vertex and -1 to the target vertex.

Clearly, the incidence matrix is unique upon fixing an order of the vertices and of the edges. For the purposes of this exposition, we will set the first entry of each column of $\mathbf{E}$ to 1 (thus the second to -1). Again referring to the example graph of FIGURE 1.1, with directions assigned and the loop in vertex 4 removed (thus the graph will have only 6 edges, as shown in FIGURE 1.2), its incidence matrix is:

$$\mathbf{E} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \\ 0 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

The following result holds.

Proposition 1.1. Let Γ be a graph with N vertices and k connected components ($1 \leq k < N$). Let $\mathbf{E}$ denote its incidence matrix. Then, it holds $\text{rank}(\mathbf{E}) = N - k$. In particular, if Γ is connected, then $\text{rank}(\mathbf{E}) = N - 1$.

We now move to the definition of Laplace matrix of a directed graph.

Definition 1.5 (Laplace matrix)**.** Given a directed graph Γ, let $\mathbf{E}$ be its incidence matrix. The N-by-N matrix $\mathbf{L}_\Gamma \equiv \mathbf{E}\mathbf{E}^T$ is called Laplace (or Laplacian) matrix of Γ.

The Laplace matrix has an interesting interpretation as a linear operator on the (N-dimensional) space of functions $f : \mathcal{V} \to \mathbb{R}$, representing the so-called discrete Laplace operator. This is the counterpart of the standard continuous Laplace operator Δ.

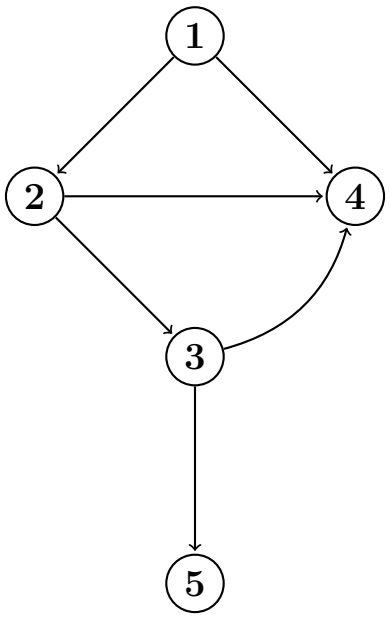


Figure 1.2: The graph of the previous example, now directed and without loops

Without entering too deeply into the details, let us investigate this fact exploiting the graph in FIGURE 1.2. Its Laplace matrix is

$$\mathbf{L}_\Gamma = \mathbf{E}\mathbf{E}^T = \begin{bmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 3 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Recall that, given a function $f : \mathcal{V} \to \mathbb{R}$, its discrete Laplacian on the graph $\Gamma = (\mathcal{V}, \mathcal{E})$ is defined as:

$$(\Delta_\Gamma f)(v) \equiv \sum_{w \in A_v} f(v) - f(w)$$

where A_v denotes the set of vertices adjacent to v (i.e. directly connected to v through an edge). The following result holds.

Proposition 1.2. Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a directed graph. The linear operator Δ_Γ on the N-dimensional space of functions $\mathbb{R}^\mathcal{V} := \{f : \mathcal{V} \to \mathbb{R}\}$ is represented by the matrix $\mathbf{L}_\Gamma$.

The Laplace matrix enjoys a number of properties. We list some of them in the following Proposition.

Proposition 1.3. The Laplace matrix $\mathbf{L}_\Gamma$ of a graph Γ has the following properties:

(a) $\mathbf{L}_\Gamma$ is symmetric and semi-positive definite.

(b) $\mathbf{L}_\Gamma$ is singular and the multiplicity of the eigenvalue 0 is equal to the number of connected components of Γ. In particular, if Γ is connected, 0 is a simple eigenvalue and the kernel of $\mathbf{L}_\Gamma$ is spanned by the vector of all ones.

(c) $\mathbf{L}_\Gamma$ is independent of the orientation of the edges in the graph.

1.2 Metric graphs

Before considering graphs as domains of differential equations, it is necessary to endow them with a metric structure. This means, simply speaking, taking into account the geometry of the edges. The starting point is the following Definition.

Definition 1.6 (Metric graph)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a connected graph. Γ is called a metric graph if

(a) a strictly positive and finite quantity l_e, called length of the edge, is assigned to each edge $e \in \mathcal{E}$;

(b) a coordinate $x^e \in [0, l_e]$, increasing along a fixed, but arbitrary, direction of the edge itself is assigned to each edge $e \in \mathcal{E}$.

The coordinates x^e in DEFINITION 1.6 are intended to work as curvilinear abscissae along each edge, thus taking into account their geometry, which in general could be that of simple differentiable curves. In the following, we will assume for simplicity that edges are straight lines joining pair of vertices.

Remark 1.3. Notice that, if M, i.e. the number of edges, is finite, then also the sum of all lengths of the edges in a graph Γ is finite. Such quantity is referred to as volume of the graph and will be denoted by $\mathbf{vol}(\Gamma)$.

A metric graph can be interpreted as a metric space in a very natural way, namely it is possible to introduce an intuitive notion of distance. It appears reasonable to define the distance between any two vertices in the graph as the length of the shortest path connecting them (recall that a metric graph is assumed to be connected). Of course, the length of a path is the sum of the length of the edges by which it is constituted. This notion of distance can be generalized to any couple of points of the graph (i.e. vertices or points even lying on different edges) using the same criterion, exploiting property (b) in DEFINITION 1.6 to measure "portions" of edges. It is straightforward to prove that this is indeed a distance.
Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a metric graph.

Definition 1.7 ($L^2(e)$ space)**.** Given an edge $e \in \mathcal{E}$, we denote by $L^2(e)$ the set of square-integrable real-valued functions $u : e \to \mathbb{R}$. We endow this space with the norm

$$\|u\|^2_{L^2(e)} = \int_e u^2(x)\,\mathrm{d}x$$

where the integral stands for a line integral along the edge e.

Functions in $L^2(e)$ can be interpreted as they were defined globally on Γ, but with support contained in e. In this way, we can define the space $L^2(\Gamma)$.

Definition 1.8 ($L^2(\Gamma)$ space)**.** The space $L^2(\Gamma)$ is defined as $\bigoplus_{e\in\mathcal{E}} L^2(e)$. This space is equipped with the norm

$$\|u\|^2_{L^2(\Gamma)} = \sum_{e\in\mathcal{E}} \|u|_e\|^2_{L^2(e)} \equiv \int_\Gamma u^2(x)\,\mathrm{d}x$$

The $L^2(\Gamma)$-inner product will be denoted by

$$(u, v)_{L^2(\Gamma)} = \int_\Gamma u(x)v(x)\,\mathrm{d}x.$$

Sobolev spaces on graphs, in the case of straight edges, can be defined with a similar approach.

Definition 1.9 ($H^1(e)$ spaces)**.** Given an edge $e \in \mathcal{E}$, we denote by $H^1(e)$ the set of functions in $L^2(e)$ whose (distributional) derivatives also are in $L^2(e)$. We endow this space with the norm

$$\|u\|^2_{H^1(e)} = \int_e \left(\frac{\mathrm{d}u}{\mathrm{d}x}(x)\right)^2 \mathrm{d}x + \int_e u^2(x)\,\mathrm{d}x = \left\|\frac{\mathrm{d}u}{\mathrm{d}x}\right\|^2_{L^2(e)} + \|u\|^2_{L^2(e)}$$

The derivative symbol stands for the directional derivative of u in the direction of the edge e.

Definition 1.10 ($H^1(\Gamma)$ space)**.** The space $H^1(\Gamma)$ is defined as $\bigoplus_{e\in\mathcal{E}} H^1(e) \cap C^0(\Gamma)$. This space is equipped with the norm

$$\|u\|^2_{H^1(\Gamma)} = \sum_{e\in\mathcal{E}} \|u|_e\|^2_{H^1(e)} = \int_\Gamma \left(\frac{\mathrm{d}u}{\mathrm{d}x}(x)\right)^2 \mathrm{d}x + \int_\Gamma u^2(x)\,\mathrm{d}x.$$

Remark 1.4. Notice that although $H^1(e) \subset C^0(e)$ (implied by Sobolev embeddings for functions of a single variable), functions in $\bigoplus_{e\in\mathcal{E}} H^1(e)$ may not be continuous. Asking for global continuity on the graph, as in DEFINITION 1.10, ensures that the value on a vertex is independent from the path followed to reach it, thus being unique.

Definition 1.11 ($L^\infty(\Gamma)$ space)**.** The space $L^\infty(\Gamma)$ is defined as the space of essentially bounded functions $f : \Gamma \to \mathbb{R}$.

1.3 Quantum graphs

One interesting application of the tools developed in the previous Sections involves the so-called quantum graphs. We stick to the basic notions needed for our purposes. An exhaustive introduction to quantum graphs can be found, for instance, in [5]. Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a metric graph. The idea behind the definition of a quantum graph is to assign to Γ an additional structure, represented by a differential operator $\mathcal{H}$, which we call Hamiltonian. In particular, we consider Hamiltonians of the following form

$$\mathcal{H}u(x) = -\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(x) + v(x)u(x)$$

where $v(x) \in L^\infty(\Gamma)$ is called potential. We recall that derivatives are taken along the edge directions. The classical case of a differential operator over a segment requires suitable boundary conditions on the vertices of the graph. In this work, we mainly deal with the so-called Neumann-Kirchhoff conditions

$$\sum_{e \in \mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v) = 0 \quad \forall\, v \in \mathcal{V}$$

where we denote by $\mathcal{E}_v$ the set of edges incident at vertex v, and the derivatives are assumed to be taken in the outgoing directions with respect to the vertex. This condition mimics the Kirchhoff current conservation law in a node of an electrical network, while also being the standard Neumann conditions for vertices with a single incident edge. Dirichlet conditions can be imposed in the classical way, typically in the context of a mixed Dirichlet-Kirchhoff-Neumann problem: given $\mathcal{V}_d = \{d_1, ..., d_r\} \subset \mathcal{V}$, being the Dirichlet vertex set, and a collection of r real constants $\{k_i\}_{i=1}^r$ we have:

$$u(d_i) = k_i \quad \forall\, i = 1, ..., r.$$

We can now give the Definition of quantum graph.

Definition 1.12 (Quantum graph)**.** A quantum graph is a triple $(\Gamma, \mathcal{H}, \mathcal{C})$, such that Γ is a metric graph, $\mathcal{H}$ an Hamiltonian and $\mathcal{C}$ denotes a set of vertex conditions.

The remainder of this report is devoted to the investigation of differential problems on quantum graphs. As long as the theoretical framework is concerned, full Kirchhoff-Neumann conditions will be imposed, while some numerical experiments are conducted also considering Dirichlet boundary data.

Chapter 2

Finite Element Metods on Graphs

2.1 Discretization of a graph and finite element spaces

From the numerical and computational standpoints, it is fundamental to understand how objects like metric graphs can be discretized in order to develop and implement algorithms. Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a metric graph with straight edges. For each edge $e \in \mathcal{E}$, let us consider a uniform mesh of $n_e \geq 2$ elements (possibily depending on e), of length $h_e = l_e n_e^{-1}$, generating $n_e - 1$ artificial vertices, which will be called discretization nodes, along the edge e. The set of such points will be denoted by $\mathcal{V}_e$. Each edge of the graph can be represented as follows

$$\{\mathcal{V}_e = \{x_j^e\}_{j=1}^{n_e-1}\} \cup \{v_a^e\} \cup \{v_b^e\}$$

namely the union of the nodes, the source and the target respectively.
In the perspective of solving differential equations on graphs, a variation of standard linear finite elements is adopted. Its structure is briefly recalled hereafter, and its derivation can be found in [7].
The method amounts to introduce an approximation of the solution of a differential equation and then to solve a linear system, as in the standard finite element method. This approximate solution belongs to a suitable finite-dimensional functional space, which is called finite element space, and the unknowns in the final algebraic problem represent its coordinates with respect to a known basis. Since the edges are straight lines, it is possible to use classical one-dimensional finite elements along the edges. Thus, standard hat basic functions are built for each discretization node on every edge. This amounts to saying that, fixed an edge e of the graph, we construct a collection of $n_e - 1$ hat basis functions ψ_j^e with support $\left[x_{j-1}^e, x_{j+1}^e\right]$ defined as

$$\psi_j^e(x^e) = \begin{cases} 1 - \dfrac{\left|x_j^e - x^e\right|}{h_e} & \text{if } x^e \in \left[x_{j-1}^e, x_{j+1}^e\right] \\ 0 & \text{otherwise} \end{cases}$$

as $j = 1, ..., n_e - 1$. The definition implies that $x_0^e = v_a^e$ and $x_{n_e}^e = v_b^e$. FIGURE 2.1 shows the behaviour of such functions along an edge of length $l_e = 50$ with $n_e = 5$ sub-edges.

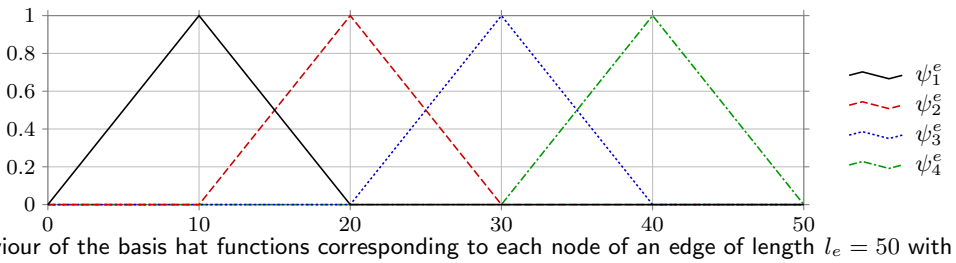


Figure 2.1: Behaviour of the basis hat functions corresponding to each node of an edge of length $l_e = 50$ with $n_e = 5$ sub-edges

As the original vertices of the graph are concerned, a similar approach is used, keeping in mind that there may be any number of adjacent vertices to them. Before describing it in detail, let us state three preliminary Definitions to avoid notational ambiguities.

Definition 2.1 (Adjacent set)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a graph, and let $v \in \mathcal{V}$. The set

$$A_v = \{w \in \mathcal{V} : (w, v) \in \mathcal{E} \text{ or } (v, w) \in \mathcal{E}\}$$

is called adjacent set of v.

Definition 2.2 (Incident set)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a graph, and let $v \in \mathcal{V}$. The set

$$\mathcal{E}_v = \{e \in \mathcal{E} : \exists\, w \in \mathcal{V} : e = (w, v) \text{ or } e = (v, w)\}$$

is called incident set of v.

Definition 2.3 (Neighbouring set)**.** Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a discretized graph, and let $v \in \mathcal{V}$. The set

$$\mathcal{W}_v = \left\{ \bigcup_{e \in \mathcal{E}_v \text{ s.t. } v_a^e = v} [v, x_1^e] \right\} \cup \left\{ \bigcup_{e \in \mathcal{E}_v \text{ s.t. } v_b^e = v} [x_{n_e - 1}^e, v] \right\}$$

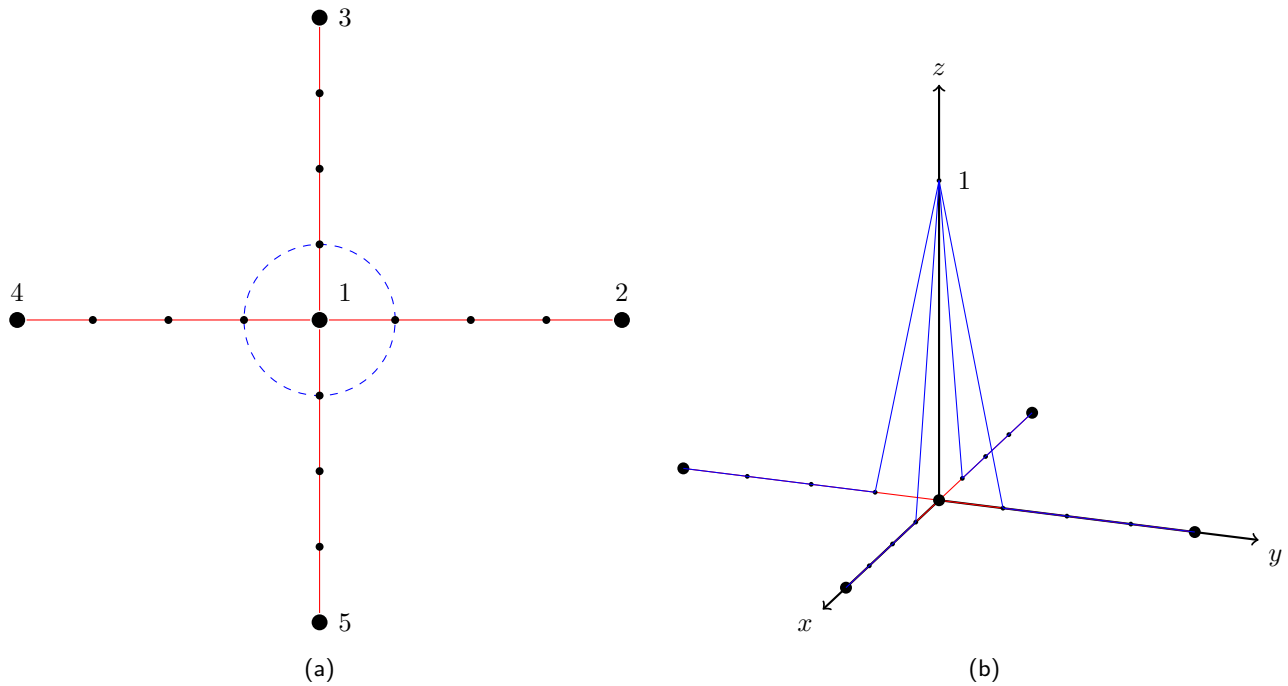
is called neighbouring set of v.

Remark 2.1. Notice that the adjacent set of a vertex v is a set of vertices, while the incident and neighbouring sets are sets of edges. The neighbouring set of a vertex v is its incident set considered on the so-called extended graph (defined later), and contains all sub-edges exiting or entering v.

The neighbouring sets $\mathcal{W}_v$ of Definition 2.3 constitute the support of the remaining basis functions, which are denoted as the family $\{\phi_v\}_{v \in \mathcal{V}}$, defined as follows, for each edge $e \in \mathcal{E}_v$:

$$\phi_v(x^e)\big|_{\mathcal{W}_v \cap e} = \begin{cases} 1 - \dfrac{|x_v^e - x^e|}{h_e} & \text{if } x \in \mathcal{W}_v \cap e \\ 0 & \text{otherwise} \end{cases}$$

where x_v^e is equal to 0 or l_e depending on v being respectively source or target of the edge e. Figure 2.2a and Figure 2.2b should better clarify these concepts on a planar star-shaped graph.



(a) (b)

Figure 2.2: (a) A star shaped graph with 5 vertices and a mesh of 4 elements for each edge. In red, the set $\mathcal{E}_v$ for the vertex 1. The set of edges inside the blue dashed circle is instead its neighbouring set $\mathcal{W}_v$. (b) In blue, the behaviour of the function ϕ_v for the vertex 1 in the graph in Figure 2.2a. Notice it reaches zero in the first node of each edge, thus being continuous on the whole graph.

The functions ψ_j^e, jointly with the functions ϕ_v, consitute the basis for the finite dimensional space in which approximations of weak solutions of differential problems can be found. In particular, fixed an edge e, the ψ_j^e span the space

$$V_h^e = \left\{ w \in H_0^1(e), w\big|_{\left[x_j^e, x_{j+1}^e\right]} \in \mathbb{P}^1, j = 0, 1, ..., n_e - 1 \right\}$$

where $H_0^1(e)$ denotes the $H^1(e)$ functions whose values at the target and source vertices are 0 and $\mathbb{P}^1$ is the set of linear functions. Consider the space

$$V_h = \left(\bigoplus_{e \in \mathcal{E}} V_h^e \right) \oplus \text{span}\left(\{\phi_v\}_{v \in \mathcal{V}} \right)$$

which is a subset of $H^1(\Gamma)$ (thus globally continuous) functions. V_h is a finite-dimensional space with dimension

$$\dim(V_h) = N + \sum_{e \in \mathcal{E}} n_e$$

which reduces to $N + n_e M$ in the case of equally meshed edges (we recall that N is the number of original vertices in the graph, and M the number of its edges). Such space is the finite element space in which approximate solutions are to be searched. All the numerical simulations shown in this report are conducted in these discrete spaces. However, as explained in CHAPTER 7, the code is implemented in order to adopt also higher-degree polynomials.

2.2 Extended graphs

Once the original graph $\Gamma = (\mathcal{V}, \mathcal{E})$ has been discretized, it is straightforward to introduce a new graph, called extended graph, whose set of vertices is formed by the union of $\mathcal{V}$ and all the discretization nodes, and whose set of edges is formed by all the sub-arcs generated by them. This new graph, let it be $\mathcal{G}$, naturally has its incidence and Laplace matrices. From the computational standpoint, all results are clearly related to the graph $\mathcal{G}$, thus we want to study what can be inferred about the original graph. In particular, we investigate whether or not the spectrum of the Laplace matrix of $\mathcal{G}$ resembles the one of the original graph. Numerical experiments seem to suggest that similarities exist in the left part of the spectra, but they are very much graph-topology dependent. Results concerning these aspects will be explained in CHAPTER 4.
An important note related to extended graphs revolves around how its vertices should be ordered. Firstly, arbitrary orders of the original edges and vertices are fixed. Once discretization nodes have been generated, the order of the vertices is determined by taking first all discretization nodes (by their order, following the direction of the edge, and by edge order), and then all original nodes. This can be better understood by looking at the incidence matrix $\mathbf{E}$ and at the sets $\mathcal{V}, \mathcal{E}$ explicitly. For instance, consider the star-shaped planar graph of FIGURE 2.2A. Before introducing the spatial discretization, the incidence matrix can be represented as follows (vertex numeration is consistent with FIGURE 2.2A):

$$\mathbf{E} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$$
$$\mathcal{E} = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5)\}$$

Once the mesh is introduced, 12 additional discretization nodes $n_1, n_2, ..., n_{12}$ are added to the vertex set $\mathcal{V}$. In this way, the extended graph vertex set is made of 17 vertices, ordered as shown in FIGURE 2.3A and FIGURE 2.3B. Of course, while the set of vertices is simply being extended, the set of edges is totally changed, since the original edges do not exist in the discretized graph. If $\mathcal{G} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, then we have:

$$\tilde{\mathcal{V}} = \left\{ \underbrace{w_1, ..., w_{12}}_{\text{artificial nodes}}, \underbrace{w_{13}, ..., w_{17}}_{\text{original nodes}} \right\}$$

where $w_i = n_i$ if $i \leq 12$, and $w_i = v_{i-12}$ otherwise. The set of edges is then composed by

$$\tilde{\mathcal{E}} = \Big\{ \underbrace{(v_1, n_1), (n_1, n_2), (n_2, n_3), (n_3, v_2)}_{\text{first edge}}, \underbrace{(v_1, n_4), (n_4, n_5), (n_5, n_6), (n_6, v_3)}_{\text{second edge}}, \underbrace{(v_1, n_7), (n_7, n_8), (n_8, n_9), (n_9, v_4)}_{\text{third edge}}, \ldots$$
$$\ldots \underbrace{(v_1, n_{10}), (n_{10}, n_{11}), (n_{11}, n_{12}), (n_{12}, v_5)}_{\text{fourth edge}} \Big\}$$

where we kept the original labeling for clarity.

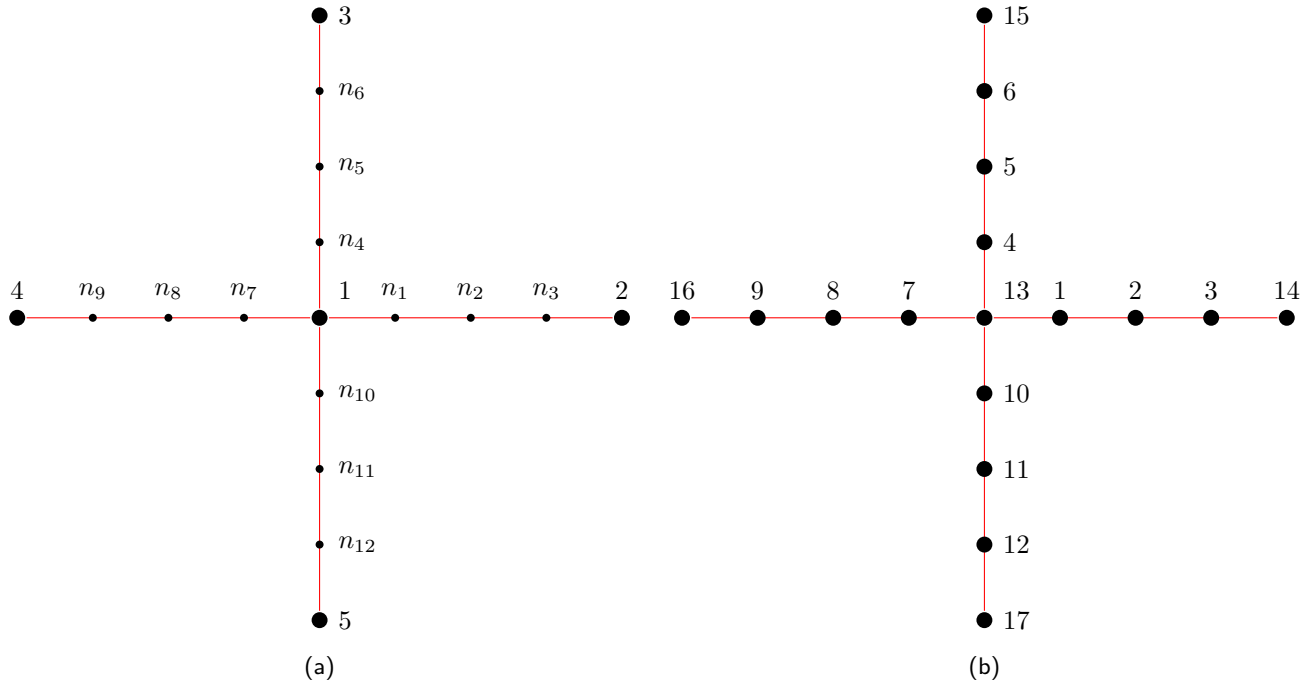


Figure 2.3: (a) The star-shaped graph discretized with 3 artificial nodes per edge. (b) The extended graph, with modified numbering: first all the artificial nodes, taken in parametrization and edge order, then all original nodes (in original order)

Consequently, the incidence matrix $\tilde{\mathbf{E}}$ of $\mathcal{G}$ is a 17-by-16 matrix as follows:

$$\tilde{\mathbf{E}} = \left[\begin{array}{cccc|cccc|cccc|cccc}
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{array}\right]$$

Horizontal lines divide the four groups of artificial nodes and the original vertices (last block of each column), whereas vertical lines divide the sub-arcs according to the edge on which they lie. Details on why this particular order is employed concern the construction of $\tilde{\mathbf{E}}$ from $\mathbf{E}$ and are not covered in this exposition, as they are quite unnecessary and cumbersome, but they can be found in [7]. Nonetheless, we remark that in this way, $\tilde{\mathbf{E}}$ has a block structure. Furthermore, the original incidence matrix, with additional null columns, appears in the lowest row of

blocks. The Laplace matrix of the extended graph is, although sparse, much larger with respect to the original one, thus it can be interesting to understand how much the space discretization affects its properties.

Chapter 3

Elliptic Differential Problems on Graphs

3.1 Generic Elliptic PDEs

3.1.1 Weak and discrete formulations

As anticipated in SECTION 1.3, we consider quantum graphs with elliptic Hamiltonians. Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a metric graph, let $v \in L^\infty(\Gamma)$ and $f : \Gamma \to \mathbb{R}$ be given. Let $\mathcal{H}$ denote the Hamiltonian of the quantum graph $(\Gamma, \mathcal{H})$, underlying Neumann-Kirchhoff conditions at any vertex in $\mathcal{V}$. The Hamiltonian acts on functions defined on Γ as described in SECTION 1.3, namely

$$\mathcal{H}u(x) = -\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(x) + v(x)u(x).$$

Let us consider the differential problem

$$\begin{cases} \mathcal{H}u = f & \text{in } \Gamma \\ \displaystyle\sum_{e \in \mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v) = 0 & \forall\, v \in \mathcal{V} \end{cases} \tag{3.1}$$

where dependence on the variable x has been omitted in the first equation. Problem (3.1) can be expressed in weak form by means of a standard procedure. Multiplying the differential equation by a test function $w \in H^1(\Gamma)$, integrating over Γ, performing integration by parts of the second-order term over each edge and injecting vertex conditions yields

$$h(u, w) = \int_\Gamma f w \,\mathrm{d}x \quad \forall\, w \in H^1(\Gamma) \tag{3.2}$$

where h is a real-valued bilinear form defined on $H^1(\Gamma) \times H^1(\Gamma)$, given by

$$h(z, w) = \sum_{e \in \mathcal{E}} \left[\int_e \frac{\mathrm{d}z}{\mathrm{d}x} \frac{\mathrm{d}w}{\mathrm{d}x} \,\mathrm{d}x + \int_e v(x) z(x) w(x) \,\mathrm{d}x \right]$$

Let us prove the following result.

Theorem 3.1. *Let $f \in L^2(\Gamma)$ and suppose it exists a real number $v_0 > 0$ be such that $v(x) > v_0$ for all $x \in \Gamma$. Then, Problem* (3.1) *possesses a unique weak solution $u \in H^1(\Gamma)$. Moreover, the following stability estimate holds:*

$$\|u\|_{H^1(\Gamma)} \leq \frac{1}{\min(1, v_0)} \|f\|_{L^2(\Gamma)}$$

Proof. The thesis follows from the Lax-Milgram Theorem, once its hypotheses are verified, namely:

- $H^1(\Gamma)$ is a Hilbert space.
- The bilinear form h is continuous. Indeed, it follows from the Cauchy-Schwarz inequality that, for any choice of z and w in $H^1(\Gamma)$:

$$h(z, w) \leq \max\left(1, \|v\|_\infty\right) \|u\|_{H^1(\Gamma)} \|w\|_{H^1(\Gamma)}$$

- The bilinear form h is coercive. Indeed, for any choice of w in $H^1(\Gamma)$:

$$h(w,w) \geq \min(1, v_0)\|w\|^2_{H^1(\Gamma)}$$

- The functional $F : H^1(\Gamma) \to \mathbb{R}$ defined as

$$F(w) = \int_\Gamma fw$$

is linear (by the properties of the line integral) and continuous (by the Cauchy-Schwarz inequality).

□

The discrete version of the problem can be obtained exploiting a standard Galerkin procedure. Let us restrict the bilinear form h to the subspace V_h, as defined in SECTION 2.1, and let us take as test functions only elements of V_h, namely consider the weak formulation (3.2) introducing a second bilinear form $h_h : V_h \times V_h \to \mathbb{R}$:

$$h_h(u_h, v_h) = \int_\Gamma f v_h \, \mathrm{d}x \quad \forall\, v_h \in V_h$$

where the unknown is now $u_h \in V_h$. Finally, recalling that V_h is finite-dimensional and considering the basis formed by the two families of functions $\{\psi^e_j\}_{j,e}$, $\{\phi_v\}_v$, let us write $v_h \in V_h$ as

$$v_h(x) = \sum_{e\in\mathcal{E}} \sum_{j=1}^{n_e-1} \alpha^e_j \psi^e_j(x) + \sum_{v\in\mathcal{V}} \beta_v \phi_v(x) \quad \forall x \in \Gamma.$$

Testing on the basis functions, one can reduce to the finite-dimensional system in the $\dim(V_h)$ constant real unknowns α^e_j, β_v:

$$\begin{cases} h_h(u_h, \psi^e_j) = \displaystyle\int_{\mathrm{supp}(\psi^e_j)} f(x)\psi^e_j(x)\,\mathrm{d}x & e \in \mathcal{E}, j = 1, ..., n_e - 1 \\ h_h(u_h, \phi_v) = \displaystyle\int_{\mathcal{W}_v} f(x)\phi_v(x)\,\mathrm{d}x & v \in \mathcal{V} \\ u_h(x) = \displaystyle\sum_{e\in\mathcal{E}} \sum_{j=1}^{n_e-1} \alpha^e_j \psi^e_j(x) + \sum_{v\in\mathcal{V}} \beta_v \phi_v(x) & \forall x \in \Gamma \end{cases}$$

or, injecting the u_h expression into h_h and exploting linearity:

$$\begin{cases} h_h(u_h, \psi^e_k) = \displaystyle\sum_{e\in\mathcal{E}} \sum_{j=1}^{n_e-1} \alpha^e_j \left\{ \int_e \frac{\mathrm{d}\psi^e_j}{\mathrm{d}x}\frac{\mathrm{d}\psi^e_k}{\mathrm{d}x}\,\mathrm{d}x + \int_e v(x)\psi^e_j(x)\psi^e_k(x)\,\mathrm{d}x \right\} + \\ \quad + \displaystyle\sum_{v\in\mathcal{V}} \beta_v \left\{ \int_{\mathcal{W}_v} \frac{\mathrm{d}\phi_v}{\mathrm{d}x}\frac{\mathrm{d}\psi^e_k}{\mathrm{d}x}\,\mathrm{d}x + \int_{\mathcal{W}_v} v(x)\phi_v(x)\psi^e_k(x)\,\mathrm{d}x \right\} = \int_{\mathrm{supp}(\psi^e_k)} f\psi^e_k\,\mathrm{d}x & e \in \mathcal{E},\ k = 1, ..., n_e - 1 \\ h_h(u_h, \phi_v) = \displaystyle\sum_{e\in\mathcal{E}} \sum_{j=1}^{n_e-1} \alpha^e_j \left\{ \int_e \frac{\mathrm{d}\psi^e_j}{\mathrm{d}x}\frac{\mathrm{d}\phi_v}{\mathrm{d}x}\,\mathrm{d}x + \int_e v(x)\psi^e_j(x)\phi_v\,\mathrm{d}x \right\} + \\ \quad + \displaystyle\sum_{v\in\mathcal{V}} \beta_v \left\{ \int_{\mathcal{W}_v} \frac{\mathrm{d}\phi_v}{\mathrm{d}x}\frac{\mathrm{d}\phi_v}{\mathrm{d}x}\,\mathrm{d}x + \int_{\mathcal{W}_v} v(x)\phi_v(x)\phi_v(x)\,\mathrm{d}x \right\} = \int_{\mathcal{W}_v} f\phi_v\,\mathrm{d}x & v \in \mathcal{V} \end{cases} \tag{3.3}$$

which can be written more compactly in vectorial form. The vector of unknowns α_j^e and β_v will be denoted as

$$\begin{bmatrix} \alpha_1^1 \\ \alpha_2^1 \\ \vdots \\ \alpha_{n_1-1}^1 \\ \alpha_1^2 \\ \vdots \\ \alpha_{n_2-1}^2 \\ \vdots \\ \alpha_1^M \\ \vdots \\ \alpha_{n_M-1}^M \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} = \begin{bmatrix} \mathbf{u}_\mathcal{E} \\ \mathbf{u}_\mathcal{V} \end{bmatrix}$$

where $\mathbf{u}_\mathcal{E}$ is the vector consisting of the α_j^e, and $\mathbf{u}_\mathcal{V}$ the one containing the β_v. We adopt a similar notation for the source terms of the discrete formulation, namely the right hand sides, distinguishing between $\mathbf{f}_\mathcal{E}$ (first set of equations) and $\mathbf{f}_\mathcal{V}$ (second set of equations). Thus, the problem reads

$$\mathbf{H} \begin{bmatrix} \mathbf{u}_\mathcal{E} \\ \mathbf{u}_\mathcal{V} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_\mathcal{E} \\ \mathbf{f}_\mathcal{V} \end{bmatrix}$$

where the square matrix $\mathbf{H}$ is called discrete Hamiltonian. As for the properties of $\mathbf{H}$, the following theorem holds (for its proof, see Theorem 3.1 in [7]).

Theorem 3.2. *The discrete Hamiltonian* $\mathbf{H}$ *has the following block structure:*

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{12}^T & \mathbf{H}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{G} \end{bmatrix} + \begin{bmatrix} \mathbf{V} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{F} \end{bmatrix} \equiv \mathbf{L} + \mathbf{M}_v.$$

In particular

(*a*) $\mathbf{H}$ *is symmetric and positive definite.*

(*b*) $\mathbf{A}$ *and* $\mathbf{V}$ *are block diagonal matrices.*

(*c*) $\mathbf{G}$ *and* $\mathbf{F}$ *are diagonal matrices.*

(*d*) $\mathbf{L}$*, called stiffness matrix, is the Laplace matrix of the extended graph.* $\mathbf{M}_v$ *is called generalized mass matrix.*

(*e*) *The entries of each block are*

$$A_{jk} = \int_e \frac{\mathrm{d}\psi_j^e}{\mathrm{d}x} \frac{\mathrm{d}\psi_k^e}{\mathrm{d}x}\,\mathrm{d}x = \begin{cases} 2/h_e & \text{if } j = k \\ -1/h_e & \text{if } |j-k| = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$B_{vk} = \int_e \frac{\mathrm{d}\psi_j^e}{\mathrm{d}x} \frac{\mathrm{d}\phi_v}{\mathrm{d}x}\,\mathrm{d}x = \begin{cases} -1/h_e & \text{if } e \in \mathcal{W}_v \cap \mathcal{E}_v \neq \varnothing \\ 0 & \text{otherwise} \end{cases}$$

$$G_{vv} = \int_{\mathcal{W}_v} \frac{\mathrm{d}\phi_v}{\mathrm{d}x} \frac{\mathrm{d}\phi_v}{\mathrm{d}x} = \sum_{e \in \mathcal{E}_v} \frac{1}{h_e}$$

$$V_{jk} = \int_e v(x)\psi_j^e(x)\psi_k^e(x)\,\mathrm{d}x$$

$$C_{vk} = \int_{\mathcal{W}_v} v(x)\phi_v(x)\psi_k^e(x)\,\mathrm{d}x$$

$$F_{vv} = \int_{\mathcal{W}_v} v(x)\phi_v(x)\phi_v(x)\,\mathrm{d}x$$

3.1.2 Solvers for linear systems

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a non-singular real-valued square matrix. Let $\mathbf{b} \in \mathbb{R}^n$ and consider the linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.4}$$

in the unknown $\mathbf{x} \in \mathbb{R}^n$. There exist many different numerical approaches to deal with Equation (3.4), as shown for example in [2]. Let us briefly present the ones that have also been employed in the C++ code presented in Chapter 7.
The *LU factorization method* consists in finding two square matrices, $\mathbf{L}, \mathbf{U}$, with the same dimensions of $\mathbf{A}$ and such that:

- $\mathbf{L}$ is lower triangular;
- $\mathbf{U}$ is upper triangular;
- $\mathbf{A} = \mathbf{L}\mathbf{U}$.

The linear system (3.4) is then solved as follows.

Algorithm 1 *LU* factorization method

```
1: procedure LU
2:     compute L and U
3:     solve for y: Ly = b
4:     solve for x: Ux = y
5:     return x
```

Being triangular, the two linear systems at lines 3 and 4 of Algorithm 1 can be solved via forward or backward substitutions. The computational cost of the procedure is dominated by the *LU* factorization itself, and takes $\frac{2}{3}n^3$ floating-point operations up to lower order terms.

Remark 3.1. The *LU* factorization of a matrix may or may not exist, in principle. Furthermore, if it exists, it may not be unique. However, if $\mathbf{A}$ is an invertible matrix and all its leading principal minors are non-zero, then it admits at least one *LU* factorization.

There are several alternatives to the *LU* factorization.
The *Conjugate Gradient (CG) method* computes the solution of the system (3.4) as a point of minimum of the quadratic form

$$\Phi(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} - \mathbf{y}^T\mathbf{b}$$

when the matrix $\mathbf{A}$ is symmetric and positive definite. In particular, given an initial guess $\mathbf{x}^0$, the initial residual $\mathbf{r}^0 = \mathbf{A}\mathbf{x}^0 - \mathbf{b}$, the method iteratively builds a descent direction forcing it to be conjugate orthogonal (i.e. orthogonal with respect to the scalar product induced by $\mathbf{A}$) to the one computed at the previous step. The stopping criterion is typically set on the norm of the residual or on the increment, as usual for iterative methods. The full scheme is shown in Algorithm 2.

Algorithm 2 Conjugate Gradient method

procedure CG
  compute $\mathbf{r}^0 = \mathbf{A}\mathbf{x}^0 - \mathbf{b}$
  $\mathbf{p}^0 = \mathbf{r}^0$
  while not converged **do**
    compute $\alpha_k = \dfrac{{\mathbf{p}^k}^T \mathbf{r}^k}{{\mathbf{p}^k}^T \mathbf{A}\mathbf{p}^k}$
    compute $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$
    compute $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k \mathbf{A}\mathbf{p}^k$
    compute $\beta_k = \dfrac{\left(\mathbf{A}\mathbf{p}^k\right)^T \mathbf{r}^{k+1}}{\left(\mathbf{A}\mathbf{p}^k\right)^T \mathbf{p}^k}$
    compute $\mathbf{p}^{k+1} = \mathbf{r}^{k+1} - \beta_k \mathbf{p}^k$
    check convergence
  return $\mathbf{x}^{k+1}$

The scalars α_k are called acceleration parameters and minimize $\Phi(\mathbf{x}^k + \alpha \mathbf{p}^k)$, while the scalars β_k ensure that the descent directions are mutually conjugate. The CG method computes the exact solution of system (3.4) in at most n steps.

Remark 3.2. The CG method is a refinement of the simpler Gradient method, in which $\mathbf{p}^k = \mathbf{r}^k$ at each step.

The CG method can be seen as part of a big family of numerical algorithms called Krylov subspace methods.

Definition 3.1 (Krylov subspace)**.** Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square matrix, $\mathbf{r} \in \mathbb{R}^n$ and $m \in \mathbb{N}$. The Krylov subspace of order m is defined as

$$K_m(\mathbf{A}; \mathbf{r}) = \text{span}\{\mathbf{r}, \mathbf{A}\mathbf{r}, \mathbf{A}^2\mathbf{r}, ..., \mathbf{A}^{m-1}\mathbf{r}\}$$

The core idea behind Krylov methods is to generate a sequence of approximated solutions $\{\mathbf{x}^k\}_{k \in \mathbb{N}}$ in such a way that

$$\mathbf{x}^k \in W_k \equiv \mathbf{x}^0 + K_k(\mathbf{A}, \mathbf{r}^0)$$

where $\mathbf{r}^0$ denotes the initial residual. The criteria adopted to generate these solutions distinguish various iterative methods. In particular, in the CG case, $\mathbf{x}^k$ is chosen as the element of the set W_k minimizing the error $\|\mathbf{x} - \mathbf{x}^k\|$ in the norm induced by $\mathbf{A}$ (which we recall is symmetric and positive definite when using the CG method). There are some alternative criteria, which are applicable in more general cases.
The *Generalized Minimum RESiduals (GMRES) method* can be employed when the matrix $\mathbf{A}$ is non-symmetric. In particular, it selects $\mathbf{x}^k$ at each iteration as the element of W_k minimizing the Euclidean norm of the residual $\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$. An orthonormal basis of K_n is constructed via a modified Gram-Schmidt-type algorithm (called the Arnoldi algorithm), since power iteration in the non-symmetric case may lead to ill conditioning. A technique often used to ease the computational costs of the GMRES routine is constituted by restarts. As the name suggests, it consists in restarting the GMRES routine after a number of steps, let them be p, to solve the modified linear system (for $\mathbf{d}$):

$$\mathbf{A}\mathbf{d} = \mathbf{r}^p = \mathbf{b} - \mathbf{A}\mathbf{x}^p$$

finding $\mathbf{x}^p + \mathbf{d}$. Of course, the method is restarted every p iterations up to convergence. This alternative version of GMRES is called GMRES(p), and it is particularly convenient memory-wise. Both the CG and the GMRES methods give the exact solutions in at most n steps. The calibration of the restart parameter (in the GMRES case) may affect the convergence properties of the GMRES method, yielding that the number of needed iterations may be greater than n. The computational cost of one step of the GMRES algorithm is not constant, and the total number of floating-point operations is expected to scale, in the worst-case scenario, as $O(n^3)$, while for the CG method, the computational complexity is expected to scale as $O(n^2)$ (as for 1-dimensional problems). Refer to [2], [6] for further details and other kinds of Krylov methods.

Remark 3.3. There exists a variant of the CG method, called *BiConjugate Gradient (BiCG) method* used to handle non-symmetric matrices. This method has poor stability properties, thus it is often employed in a stabilized version, called *BiConjugate Gradient STABilized (BiCGSTAB) method*, which is basically obtained alternating one step of BiCG and one of GMRES(1).

3.2 The eigenvalue problem

3.2.1 Weak and discrete formulations

Let $(\Gamma, \mathcal{H})$ be a quantum graph as in the previous Section with a given potential $v \in L^\infty(\Gamma)$. Consider the eigenvalue problem

$$\begin{cases} \mathcal{H}u = \lambda u & \text{in } \Gamma \\ \displaystyle\sum_{e \in \mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v) = 0 & \forall\, v \in \mathcal{V} \end{cases} \tag{3.5}$$

where the unknowns are the eigencouples (λ, u).
Following the same reasoning of the previous Section, it is possible to formulate weak and discrete formulations of Equation (3.5). In particular, the former is similar to the one expressed by Equation (3.2): it has the same left hand side, whereas the right hand side now also depends on the eigenfunction u, namely:

$$h(u, w) = \lambda \int_\Gamma u(x)w(x)\,\mathrm{d}x$$

where h is the same bilinear form of the previous Section. Equal considerations to derive the discrete formulation:

$$\begin{cases} h_h(u_h, \psi_k^e) = \lambda f_h(u_h, \psi_k^e) & e \in \mathcal{E},\ k = 1, ..., n_e - 1 \\ h_h(u_h, \phi_v) = \lambda f_h(u_h, \phi_v) & v \in \mathcal{V} \\ u_h(x) = \displaystyle\sum_{e \in \mathcal{E}} \sum_{j=1}^{n_e - 1} \alpha_j^e \psi_j^e(x) + \sum_{v \in \mathcal{V}} \beta_v \phi_v(x) & \forall x \in \Gamma \end{cases} \tag{3.6}$$

where we recall that

$$h_h(z, w) = \sum_{e \in \mathcal{E}} \left[\int_e \frac{\mathrm{d}z}{\mathrm{d}x} \frac{\mathrm{d}w}{\mathrm{d}x}\,\mathrm{d}x + \int_e v(x)z(x)w(x)\,\mathrm{d}x \right]$$

and the new real-valued bilinear form

$$f_h(z, w) = \lambda \int_\Gamma z(x)w(x)\,\mathrm{d}x$$

are both defined on $V_h \times V_h$. Injecting the expression of u_h into the two groups of equation it is straightforward to derive the generalized eigenvalue problem:

$$\mathbf{Hu} = \lambda \mathbf{Mu}$$

where $\mathbf{H}$ denotes the discrete Hamiltonian, $\mathbf{u}$ denotes the unknown eigenvector and $\mathbf{M}$, following the notation of Theorem 3.2, is the mass matrix.

Remark 3.4. If, instead of (3.5), a "generalized eigenfunction problem" is considered, namely, given a function $p \in L^\infty(\Gamma)$

$$\begin{cases} \mathcal{H}u = \lambda p u & \text{in } \Gamma \\ \displaystyle\sum_{e \in \mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v) = 0 & \forall\, v \in \mathcal{V} \end{cases} \tag{3.7}$$

then the discrete formulation reads

$$\mathbf{Hu} = \lambda \mathbf{M}_p \mathbf{u}.$$

Remark 3.5. It is clear that non-trivial eigenvalue problems are never well-posed problems. In fact, they admit infinitely many solutions, which form a vector space called eigenspace associated to the eigenvalue λ. Spectral theory for compact operators for elliptic Hamiltonians gives that their set of eigenvalues is discrete (i.e. finite or countable) and the corresponding eigenspaces are always finite-dimensional. The aim is thus to find a basis of each eigenspace.

Remark 3.6. Dirichlet conditions in eigenvalue problems are interesting only if homogeneous. Indeed, the sum of two eigenfunctions is still expected to be an eigenfunction.

3.2.2 Solvers for generalized eigenvalue problems

Among a huge literature on numerical methods to tackle generalized eigenvalue problems, two methods are presented hereafter, as they are also employed in the C++ implementation described in CHAPTER 7.
Let $\mathbf{A}, \mathbf{B}$ be given square real matrices of dimension n and consider the generalized eigenvalue problem in the unknown eigencouples $(\lambda, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^n$

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{B}\mathbf{x}$$

The *implicit QR algorithm* is used on standard (i.e. not generalized) eigenvalue problems. Assuming $\mathbf{B}$ is invertible, one can restate the problem as

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda \mathbf{x}.$$

Remark 3.7. It can be shown that indeed the case of $\mathbf{B}$ invertible is the most interesting, as there exist n eigenvalues (counted with multiplicity) if and only if $\text{rank}(\mathbf{B}) = n$, while if $\det(\mathbf{B}) = 0$ there may be a finite, infinite or even empty set of eigenvalues.

Let, for ease of notation, $\mathbf{B}^{-1}\mathbf{A} = \mathbf{C}$. The QR algorithm has its foundations in the QR factorization. Nonetheless, the implicit version does never compute it explicitly. Furthermore, $\mathbf{C}$ is first reduced to upper Hessenberg form (i.e. all elements below the first subdiagonal are 0), which costs about $\frac{10}{3}n^3$ floating-point operations (ignoring lower order terms). The full procedure is shown hereafter.

Algorithm 3 Implicit QR algorithm

```
1: reduce C= C^0 to upper Hessenberg form
2: procedure IMPLICITQR
3:     while not converged do
4:         compute C^{k+1} = R^k Q^k
5:         check convergence
6:     return C^{k+1}
```

Remark 3.8. It is important to stress that the implicit version of the QR algorithm does not compute the QR factorization explicitly, but exploits a result known as Implicit Q Theorem to construct $\mathbf{Q}$. Further details can be found in [6].

Working with Hessenberg matrices is convenient since it can be shown that a QR step preserves their form and has a computational cost of $O(n^2)$ instead of $O(n^3)$. Thus, globally the implicit QR algorithm has its computational costs scaling as $O(n^3)$. The sequence of matrices is expected to converge to a triangular matrix (whose eigenvalues are on the main diagonal). The columns of $\mathbf{Q}$ are the searched eigenvectors.
A second method designed specifically for generalized eigenvalue problems is a modification of the QR algorithm, called *QZ algorithm*. Let us consider the generalized eigenvalue problem in its original form. The matrix $\mathbf{A}$ is reduced to upper Hessenberg form, as in the previous case, while $\mathbf{B}$ is reduced to Schur form (upper triangular). Then, the aim is to obtain the generalized Schur decomposition of $\mathbf{A}$, transforming it to upper quasi-triangular (i.e. block upper triangular with 2-by-2 blocks on the diagonal) while $\mathbf{B}$ is kept in Schur form.

Theorem 3.3 (generalized Schur decomposition)**.** *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$. Then, there exist orthogonal matrices $\boldsymbol{Q}, \boldsymbol{Z} \in \mathbb{R}^{n\times n}$ such that $\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Z}$ is upper quasi-triangular and $\boldsymbol{Q}^T\boldsymbol{B}\boldsymbol{Z}$ is upper triangular.*

Eigenvalues and eigenvectors can be computed from the triangular systems. Matrices $\mathbf{Q}$ and $\mathbf{Z}$ may be needed and can be computed as well. The total computational cost is $62n^3$ floating-point operations, divided as follows:

$$62n^3 = \underbrace{30n^3}_{\text{Schur decomposition}} + \underbrace{16n^3}_{\mathbf{Q}\text{ computation}} + \underbrace{16n^3}_{\mathbf{Z}\text{ computation}}$$

A full derivation and detailed description of the QZ algorithm can be found in [4].

Chapter 4

Numerical Experiments

This Chapter is entirely devoted to numerical simulations of problems on quantum graphs. All the tests and results presented in this Chapter are obtained with the usage of the C++ code described in due detail in Chapter 7. For visualization reasons, all simulations are conducted on planar graphs, although the code is also able to deal with 3D graphs.

4.1 Elliptic PDEs

Firstly, some validation experiments are performed in order to test the code. To this end, elliptic equations with known exact solutions are considered.

4.1.1 Test 1: constant function

Let Γ be a four-pointed star, lying on the xy plane, represented in Figure 4.1, with 5 vertices and 4 edges of length 1, whose directions coincide with the coordinate axes.

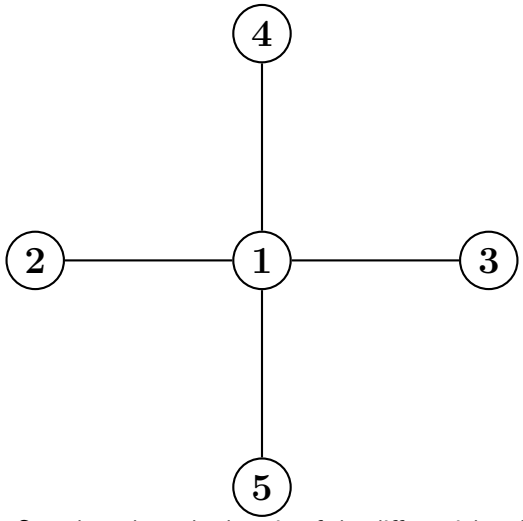


Figure 4.1: Star-shaped graph, domain of the differential problem (4.1)

Let us consider the problem

$$\begin{cases} -\Delta u + (x^2+y^2+1)u = 5(x^2+y^2+1) & \text{in } \Gamma \\ \displaystyle\sum_{e\in\mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v) = 0 & \forall\, v \in \mathcal{V} \end{cases} \tag{4.1}$$

which has the exact solution

$$u(x,y) \equiv 5.$$

In this simple case, the numerical algorithms are expected to return the exact solution. This is indeed the case, as shown in the following Figure. A mesh with 100 elements per edge is used to discretize the graph, and the *LU* factorization method is employed as solver.

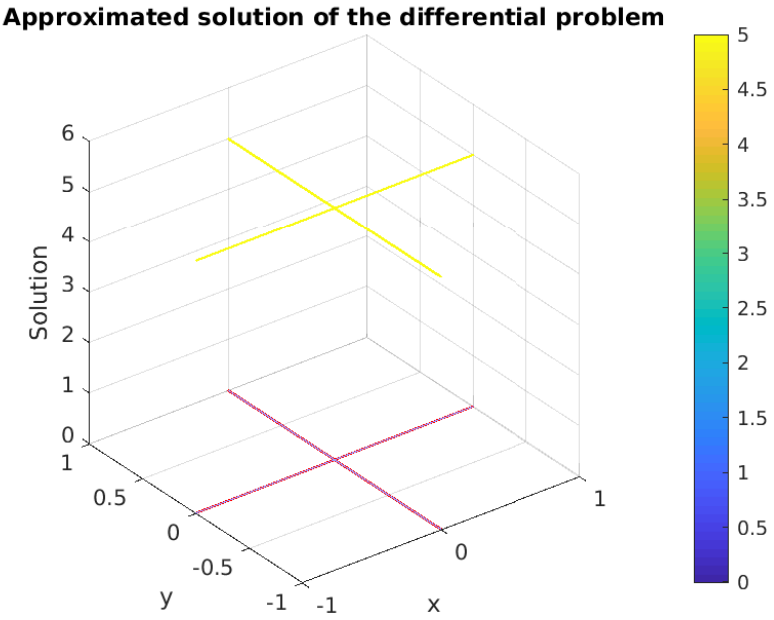


Figure 4.2: Plot of the numerical solution of Problem (4.1). In red, the computational domain.

4.1.2 Test 2: linear function

On the domain represented in FIGURE 4.1, let us consider the mixed Dirichlet-Neumann-Kirchhoff problem

$$\begin{cases} -\Delta u + (x^2+y^2+1)u = (x+y)(x^2+y^2+1) & \text{in } \Gamma \\ \displaystyle\sum_{e\in\mathcal{E}_v} \frac{\mathrm{d}u}{\mathrm{d}x^e}(v_1) = 0 \\ u(v_2) = -1 \\ u(v_3) = 1 \\ u(v_4) = 1 \\ u(v_5) = -1 \end{cases} \tag{4.2}$$

where v_i corresponds to vertex labeled i in FIGURE 4.1. Problem (4.2) has the exact solution

$$u(x,y) = x + y$$

Also in this second case, since the solution belongs to the finite element space, the numerical algorithms are expected to return the exact solution. This is indeed the case, as shown in the following Figure. A mesh with 100 elements per edge is used to discretize the graph, and the *LU* factorization method is employed as solver.

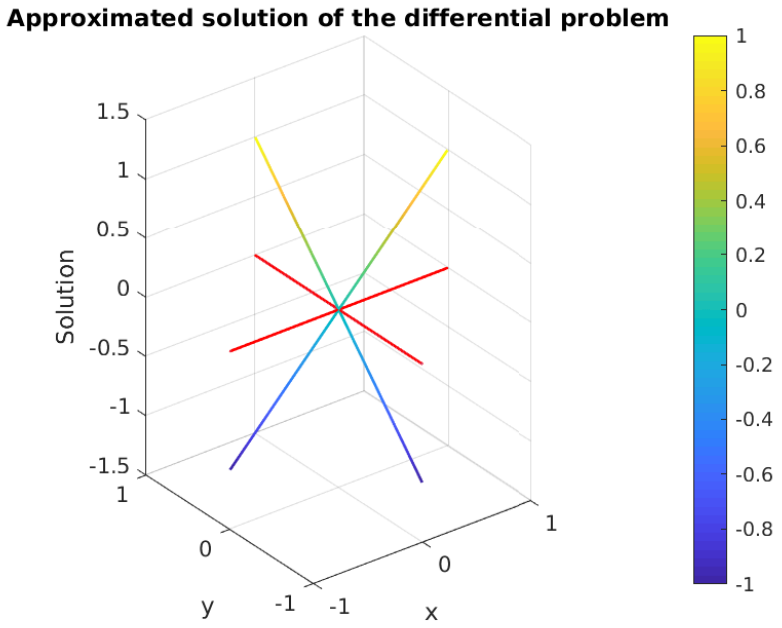


Figure 4.3: Plot of the numerical solution of Problem (4.2). In red, the computational domain.

4.1.3 Test 3: function of a single variable (x)

On the domain represented in FIGURE 4.1, let us consider the fully homogeneous Dirichlet problem

$$\begin{cases} -\Delta u + u = (4\pi^2 + 1)\sin(2\pi x) & \text{in } \Gamma \\ u(v) = 0 & \forall\, v \in \mathcal{V} \end{cases} \tag{4.3}$$

Problem (4.3) has the exact solution

$$u(x, y) = \sin(2\pi x)$$

In this case, the numerical algorithms are not expected to return the exact solution. However, with linear finite elements, it holds convergence of order 1 with respect to the mesh thickness. The plot of the discrete solution is shown in the following Figure.

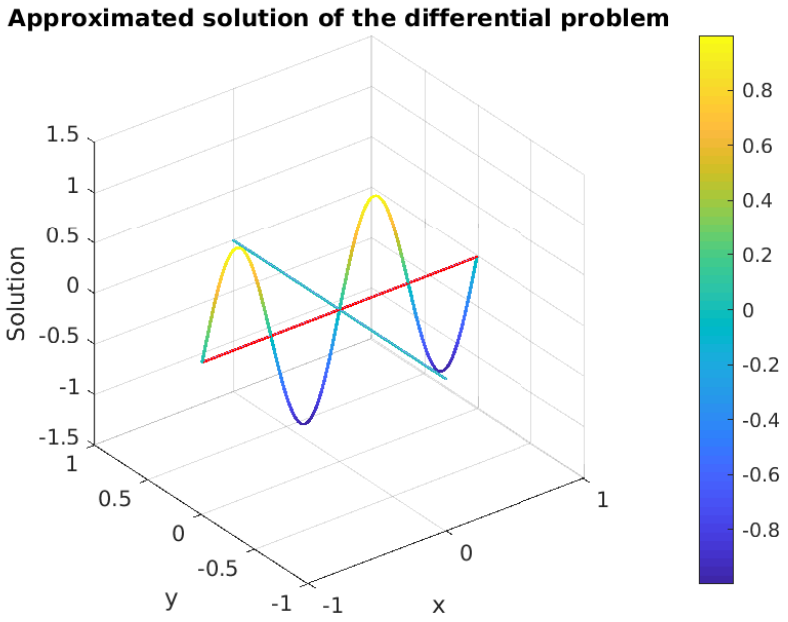


Figure 4.4: Plot of the numerical solution of Problem (4.3). In red, the computational domain.

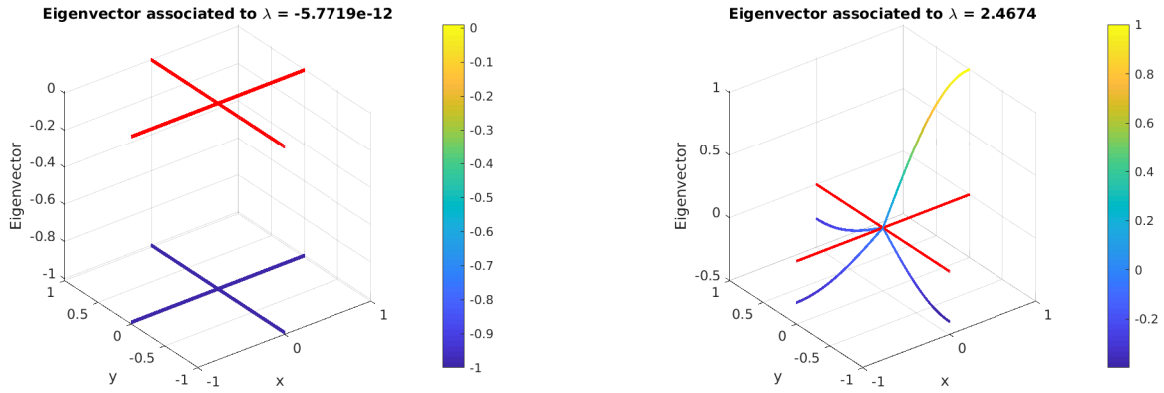
Remark 4.1. Notice that the discrete solution is correctly globally continuous, constant along the y direction with value $u(0, y) = 0$.

4.2 Eigenvalue problems

All the following numerical experiments concern the eigenvalue problem (3.5) for the Laplacian operator $-\Delta \equiv -\frac{\mathrm{d}^2}{\mathrm{d}x^2}$ (i.e. $v(x) \equiv 0$). Three different graph topologies are considered, in order to draw comparisons with the results obtained in [7].

4.2.1 Test 1: four-pointed star

Let us consider the four-pointed star shown in Figure 4.1. Hereafter are the first approximated six eigencouples of the Laplacian operator on Γ (assuming full homogeneous Neumann-Kirchhoff conditions), computed using a mesh with $n_e = 100$ elements per edge, employing the implicit QR algorithm.



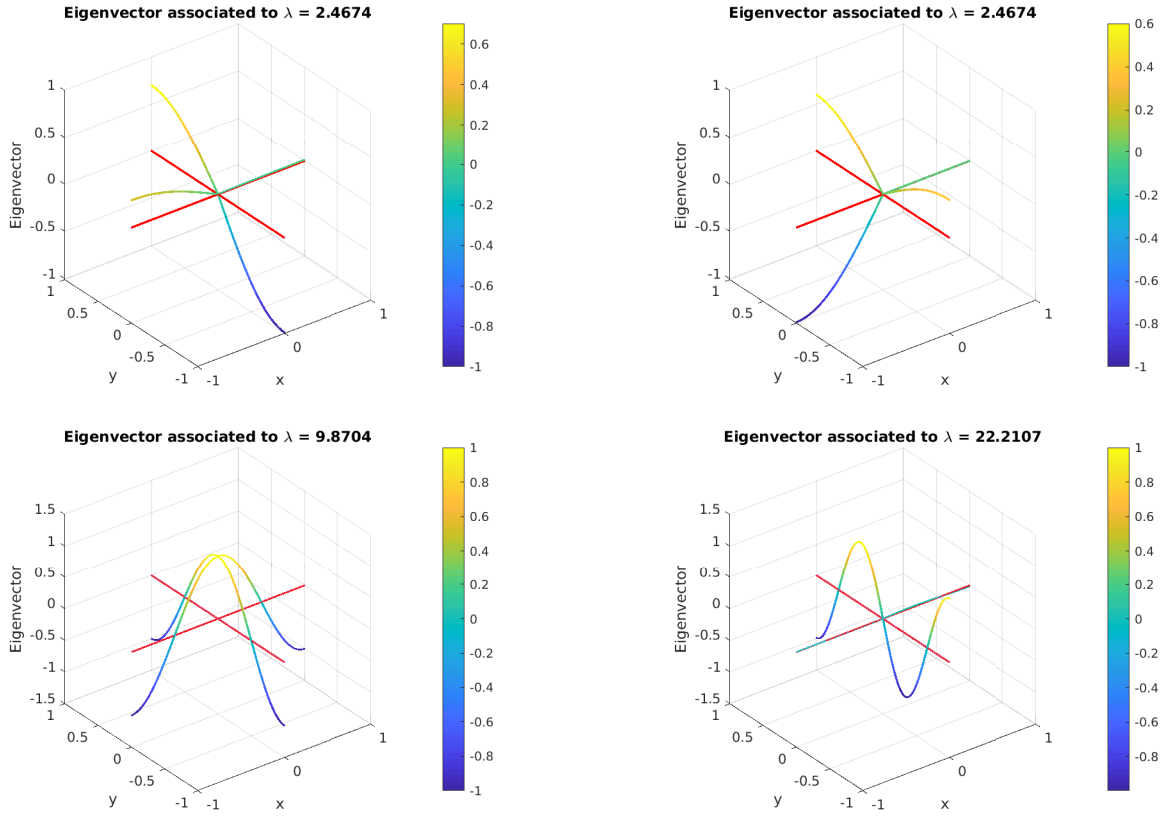


Figure 4.5: First six eigencouples of the Laplacian operator on a star-shaped graph (100 sub-arcs per edge). In red, the domain of the problem. Eigenvectors are normalized.

Each subplot of FIGURE 4.5 reports, in red, the domain on which eigenfunctions are defined. As expected from theory, all eigenfunctions are globally continuous. These results seem very much in accordance with what is reported in [7], up to normalizations and order of eigenvectors associated to multiple eigenvalues.

4.2.2 Test 2: graphene

The second graph topology resembles the molecular structure of graphene and has 12 vertices and 13 edges of unitary length.

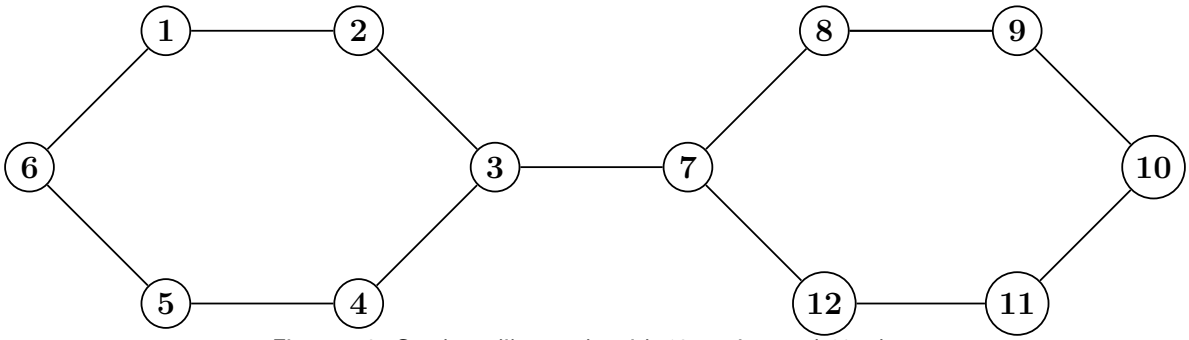
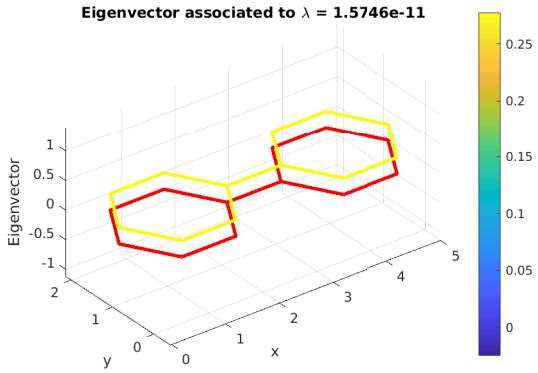
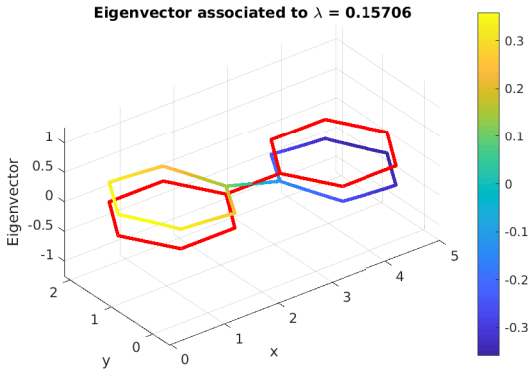
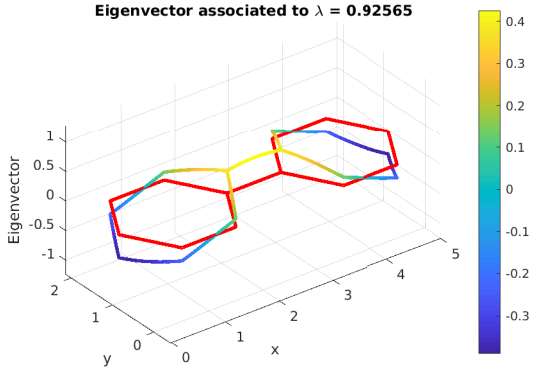


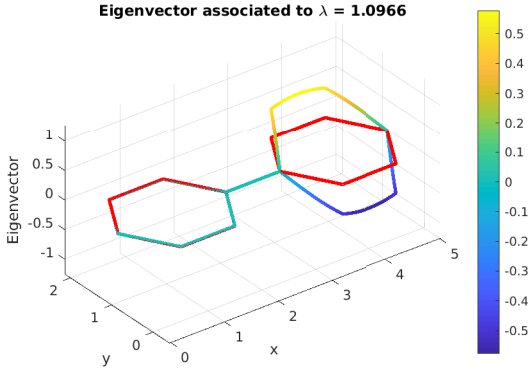
Figure 4.6: Graphene-like graph, with 12 vertices and 13 edges

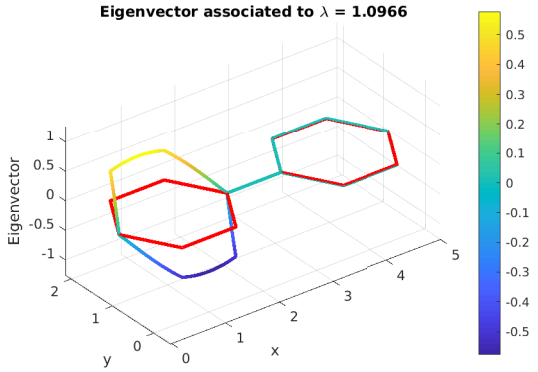
In the next Figures the first ten eigencouples are displayed, in the same fashion of the previous Subsection. Again, 100 sub-arcs per edge are introduced to discretize the graph, and the implicit QR algorithm is employed as solving routine. For visualization reasons, eigenvectors are rescaled by a 10 factor.

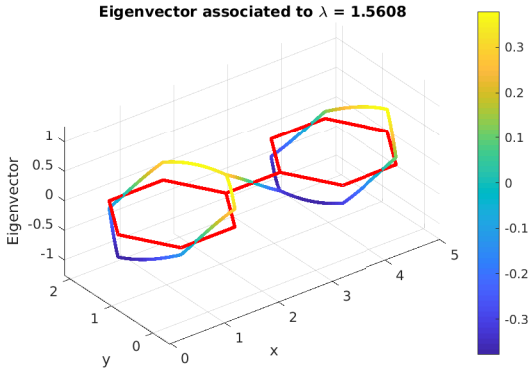

Eigenvector associated to λ = 1.5746e-11
Eigenvector
x
y

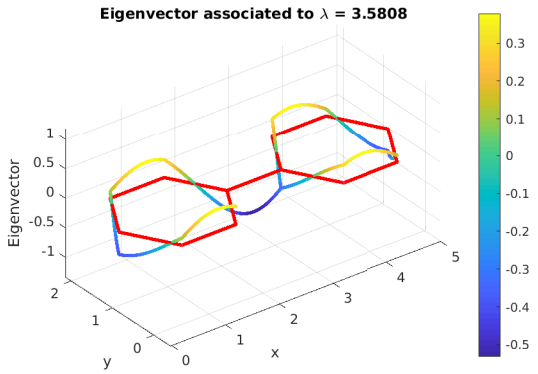

Eigenvector associated to λ = 0.15706
Eigenvector
x
y

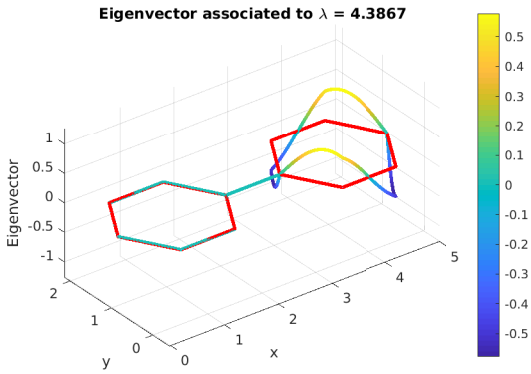

Eigenvector associated to λ = 0.92565
Eigenvector
x
y


Eigenvector associated to λ = 1.0966
Eigenvector
x
y


Eigenvector associated to λ = 1.0966
Eigenvector
x
y


Eigenvector associated to λ = 1.5608
Eigenvector
x
y


Eigenvector associated to λ = 3.5808
Eigenvector
x
y


Eigenvector associated to λ = 4.3867
Eigenvector
x
y

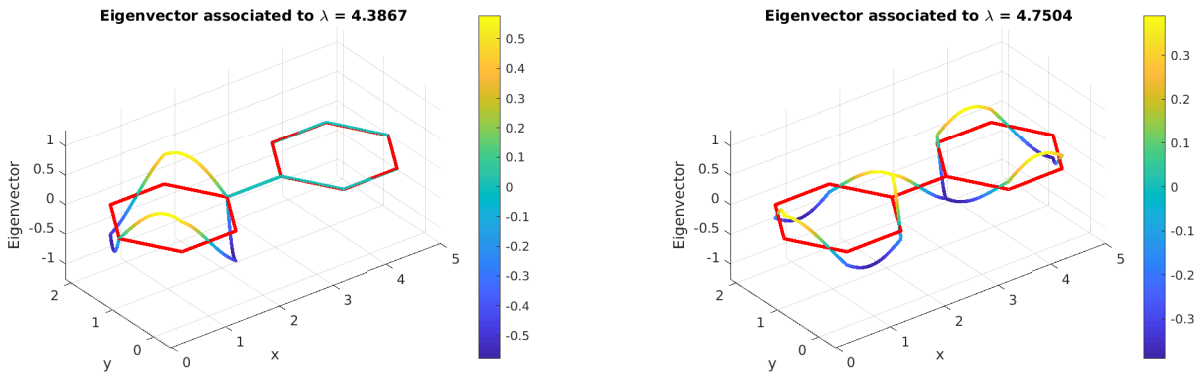


Figure 4.7: First ten eigencouples of the Laplacian operator on a graphene-like graph (100 sub-arcs per edge). In red, the domain of the problem. Eigenvectors are rescaled by a 10 factor.

Also in this case, the results are in good agreement with literature results, such as [7].

4.2.3 Test 3: tree

The third topology of interest (which will be further analyzed in CHAPTER 5) is the tree one. In particular, we consider trees of the kind shown in Figure 4.8, i.e. an edge followed by a series of bifurcations.

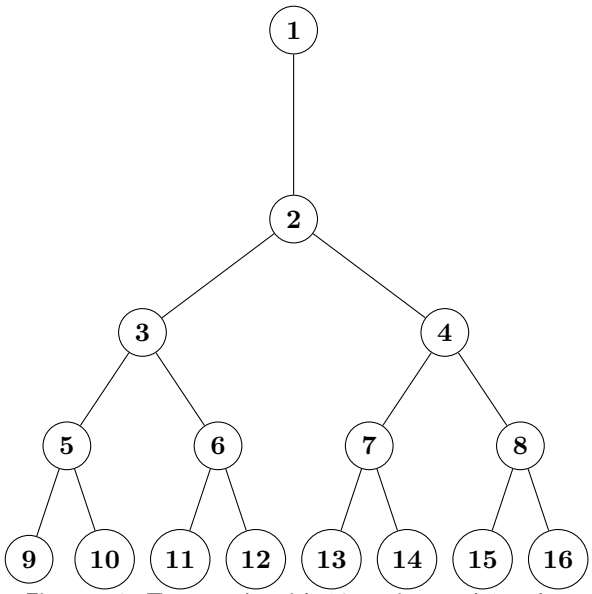
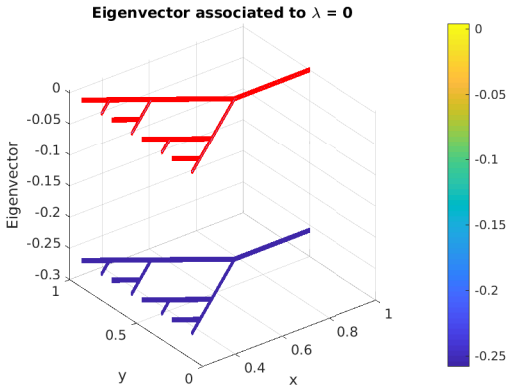
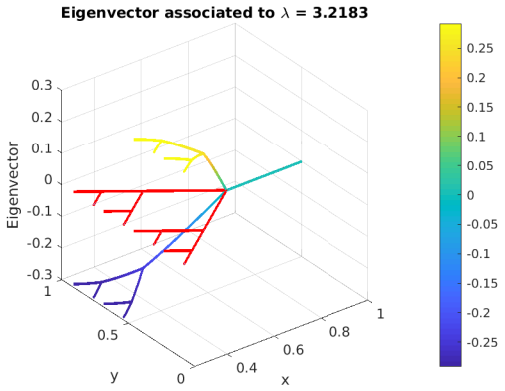
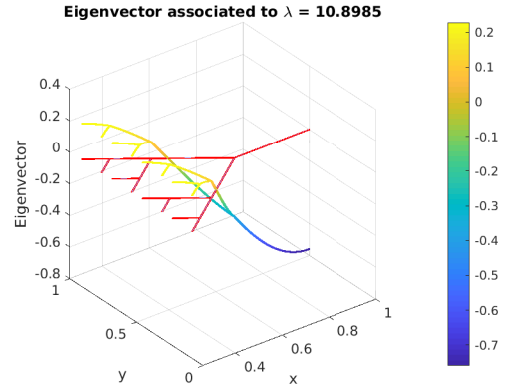


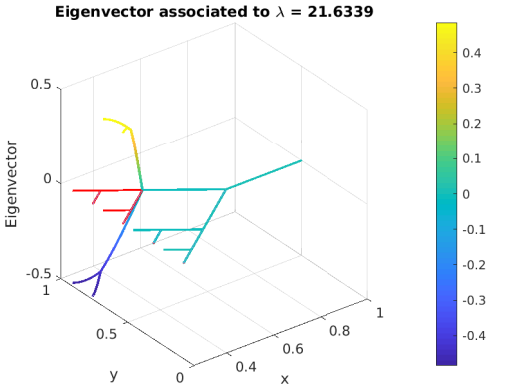
Figure 4.8: Tree graph, with 16 vertices and 15 edges

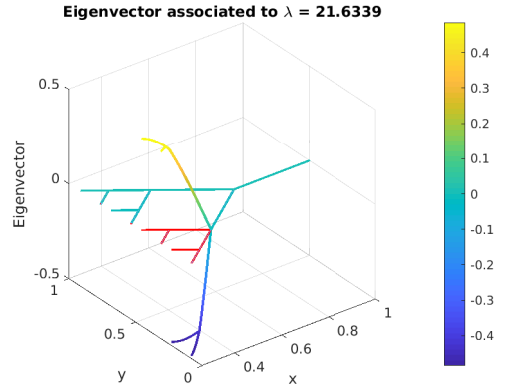
The first ten approximated eigencouples are displayed hereafter. As in the previous cases, 100 sub-arcs have been generated on each branch to discretize the graph, and the implicit QR algorithm is employed as solver. For visualization reasons, eigenvectors are rescaled by a 10 factor.

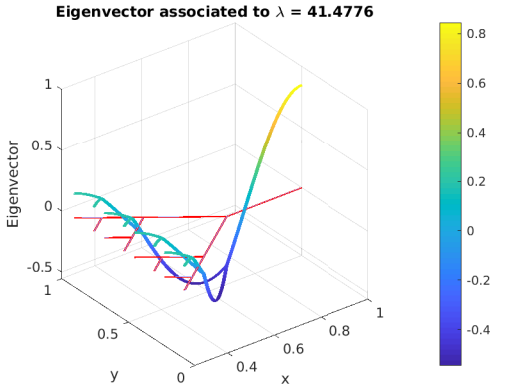

Eigenvector associated to λ = 0
Eigenvector
x
y

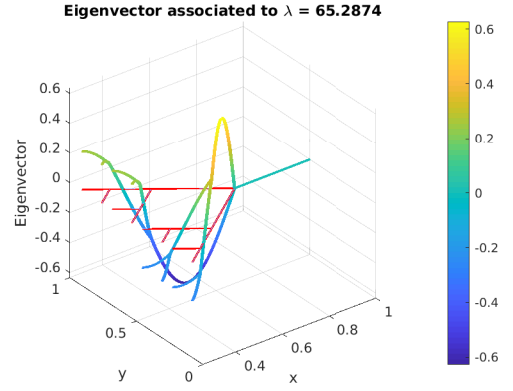

Eigenvector associated to λ = 3.2183
Eigenvector
x
y

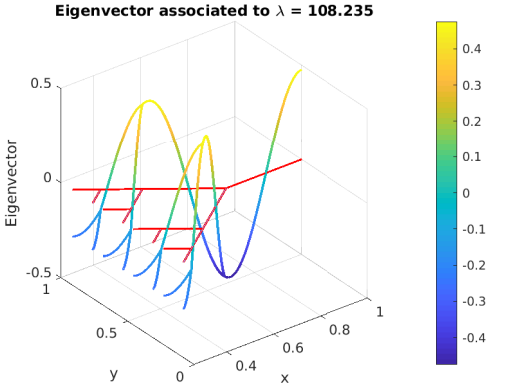

Eigenvector associated to λ = 10.8985
Eigenvector
x
y


Eigenvector associated to λ = 21.6339
Eigenvector
x
y


Eigenvector associated to λ = 21.6339
Eigenvector
x
y


Eigenvector associated to λ = 41.4776
Eigenvector
x
y


Eigenvector associated to λ = 65.2874
Eigenvector
x
y


Eigenvector associated to λ = 108.235
Eigenvector
x
y

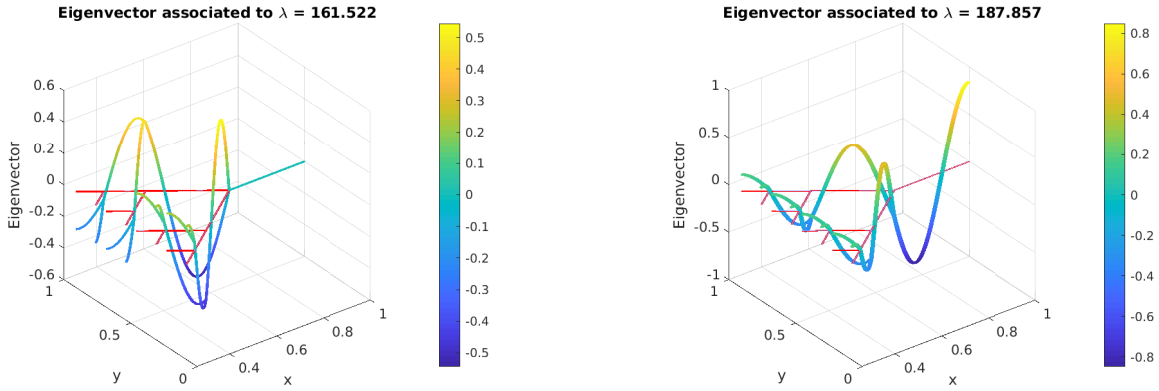


Figure 4.9: First ten eigencouples of the Laplacian operator on a tree-like graph (100 sub-arcs per edge). In red, the domain of the problem. Eigenvectors are rescaled by a 10 factor.

Remark 4.2. These results strongly resemble what is obtained in [7]. Notice that they are consistent with what is theoretically predicted: eigenfunctions present, as the eigenvalue index increases, a progressively more oscillatory behaviour, reflecting the fact that the Laplacian operator is unbounded. These considerations also apply to results obtained in SUBSECTIONS 4.2.1 and 4.2.2.

4.2.4 Laplace matrix spectral comparison

As anticipated in SECTION 2.2, given a graph Γ and its extended version $\mathcal{G}$, it is appropriate to assess whether or not the spectral properties of the Laplace matrix of $\mathcal{G}$ resemble the ones of the Laplace matrix related to Γ. To avoid ambiguities, in the following we refer to the two matrices as extended Laplace matrix and combinatorial Laplace matrix respectively. In order to perform this comparison, the eigenvalues of the combinatorial matrix are computed: this is done through the `eig()` routine of MATLAB. Of course, being the combinatorial Laplace matrix smaller with respect to the extended one, the spectral comparison makes sense only between the eigenvalues of the former and the first N eigenvalues of the latter. Here N denotes the number of vertices in the original graph (we recall that the combinatorial Laplace matrix is a N-by-N matrix). The combinatorial Laplace matrix is built through a custom MATLAB routine.
The eigenvalue comparison is drawn for a number of different graph topologies, with particular attention to tree graphs. In particular:

1. star-shaped graph, as in SUBSECTION 4.2.1;
2. graphene-like graph, as in SUBSECTION 4.2.2;

and, specifically concerning tree graphs,

3. constant edge length (unitary), decreasing bifurcation angle (decaying as $\frac{1}{n}$, where n stands for the bifurcation level);
4. constant bifurcation angle ($\frac{\pi}{4}$ or $\frac{\pi}{6}$), decreasing edge length (decaying as $\frac{1}{n}$, or $\frac{1}{n^2}$).
5. decreasing edge length and bifurcation angle (angles decay as $\frac{1}{n}$, edge length as $\frac{1}{n}$ or $\frac{1}{n^2}$).

Finally, a more complex graph is considered, namely

6. Barabási-Albert graph, built according to the Barabási-Albert model [1] using the `CONTEST` tool of MATLAB.

Results are shown in the following Figures, plotting eigenvalues against their index. The exact numerical values can be checked in Appendix B. For the star and graphene cases, each edge is divided in 100 sub-arcs. For each of the tree graph cases, the mesh is computed imposing a spatial step of $\frac{1}{46}$ on each edge. For the Barabási-Albert case, a spatial step of 0.024 is imposed. In all cases the QZ algorithm is employed.

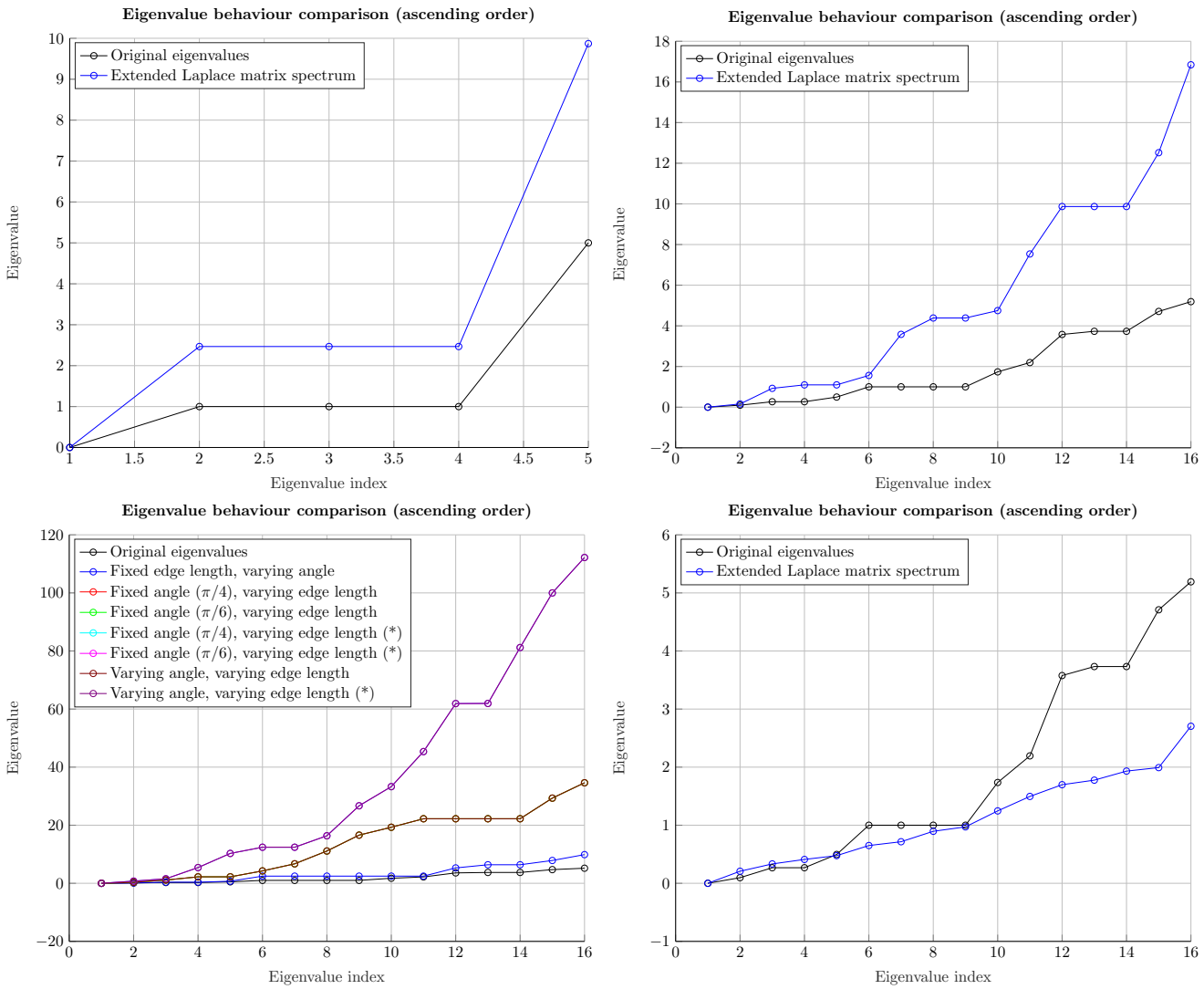


Figure 4.10: Comparison between the spectra of the combinatorial and extended Laplace matrices, varying the underlying graph topology. From top to bottom, left to right: star, graphene, trees, Barabási-Albert. (*) legend entries refer to $\frac{1}{n^2}$ edge length decay.

Remark 4.3. The presence of symmetries in the graph topology seem to cause the existence of multiple eigenvalues. This is also evident from the plots of the eigenvectors presented in the previous Subsections.

Remark 4.4. In FIGURE 4.10, in the plot related to the tree graph (bottom-left), some eigenvalue trends are overlapped. In particular, other than the black line representing the original eigenvalues, there are (from the bottom up):

1. the fixed edge length, varying angle case (dark blue; second legend entry);
2. the fixed angle ($\frac{\pi}{4}$ and $\frac{\pi}{6}$), varying edge length and varying angle, varying edge length cases (brown, overlapped to green and red; third, fourth and seventh legend entries);
3. the fixed angle ($\frac{\pi}{4}$ and $\frac{\pi}{6}$), varying edge length and varying angle, varying edge length cases with $\frac{1}{n^2}$ edge length decay (purple, overlapped to cyan and magenta; fifth, sixth and eighth legend entries).

Consequently, numerical experiments seem to suggest that, the amplitude of the bifurcation angle does not play a significant role in affecting the spectral properties of the extended Laplace matrix. Indeed, the plots are basically overlapped according to the type of edge length decay (null, linear or quadratic decay).

Remark 4.5. FIGURE 4.10, jointly with the data reported in Appendix B, shows that although the eigenvalues of the extended Laplace matrix are generally larger in value with respect to the ones of the combinatorial one,

in several cases the eigenvalue multiplicity seems to be preserved. The exceptions seem to be the graphs with non-uniform edge lengths (i.e. almost all tree topologies and the Barabási-Albert graph).

Remark 4.6. The plot related to the Barabási-Albert graph (bottom-right in FIGURE 4.10) strongly differs in behaviour with respect to all other cases, confirming that the properties stated in the previous remarks do not hold in general.

4.3 Computational complexity

It is interesting to verify how the computational complexity scales with respect to the parameters of the problem. As far as iterative methods are concerned, the aim is to verify theoretical results concerning how the number of operations needed to achieve convergence scale as the mesh is refined. Unfortunately, for the QR and QZ methods, there is no access through the available interfaces to the number of iterations needed to achieve convergence. However, through the `<chrono>` header, it is possible to measure computational times. Therefore, execution times have been plotted against the number of total degrees of freedom for the test cases illustrated in SUBSECTIONS 4.2.1 and 4.2.2. The analysis is conducted on the graphene-like topology shown in FIGURE 4.6, using an increasing number of discretization nodes per edge. In particular, both the reference problems have the potential term $v(x) \equiv 1$ and the elliptic problem has the source term $f(x, y) = x+y$ (written in Cartesian form). In both cases, full homogeneous Neumann-Kirchhoff conditions are assumed. Results are shown hereafter.

N_{dofs}	**QR**	**QZ**
194	0.964	0.264
389	5.478	1.101
649	24.226	6.343
1299	200.446	55.498
1949	692.083	205.329
2599	1728.62	535.532

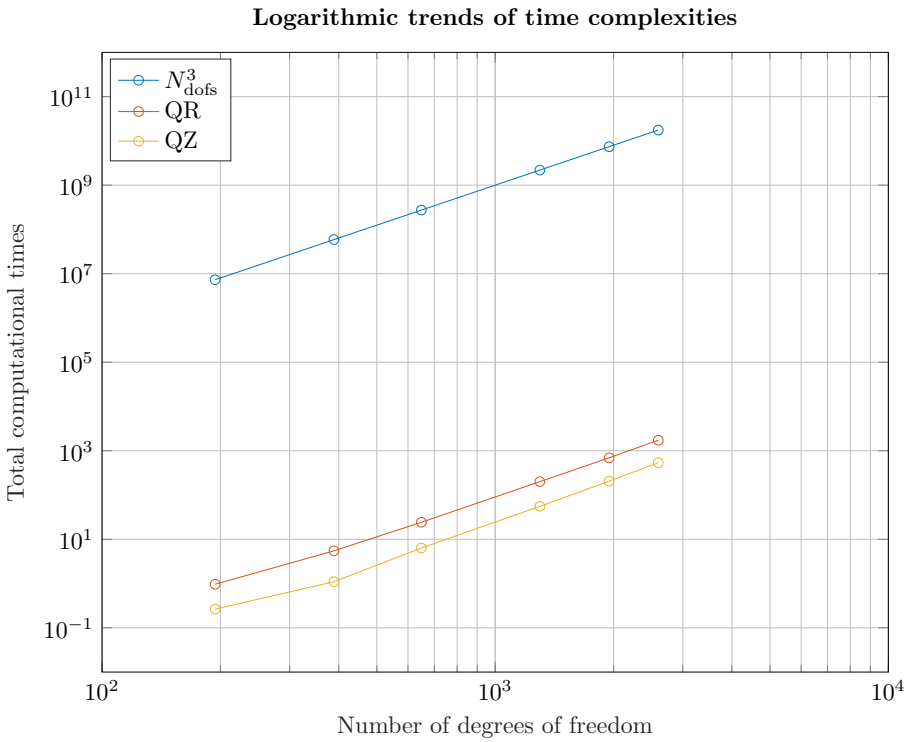


Table 4.1: Computational times for the QR and QZ methods, truncated at the third decimal digit (left). Logarithmic plot of data (right). Times are measured in seconds.

Remark 4.7. Notice that both trends are in agreement with the theoretical complexities. However, the QR algorithm should in general require a lower number of operations with respect to the QZ one. This discrepancy is due to the fact that, in the QR algorithm applied to generalized eigenvalue problems, the computation of $\mathbf{B}^{-1}\mathbf{A}$ is required. The complexity of this operation is dominated by the matrix multiplication.

N_{dofs}	**GMRES**	**QMR**
194	0.246	0.0740
389	0.831	0.211
649	3.103	0.386
1299	21.182	1.192
1949	70.236	2.555
2599	149.164	4.358

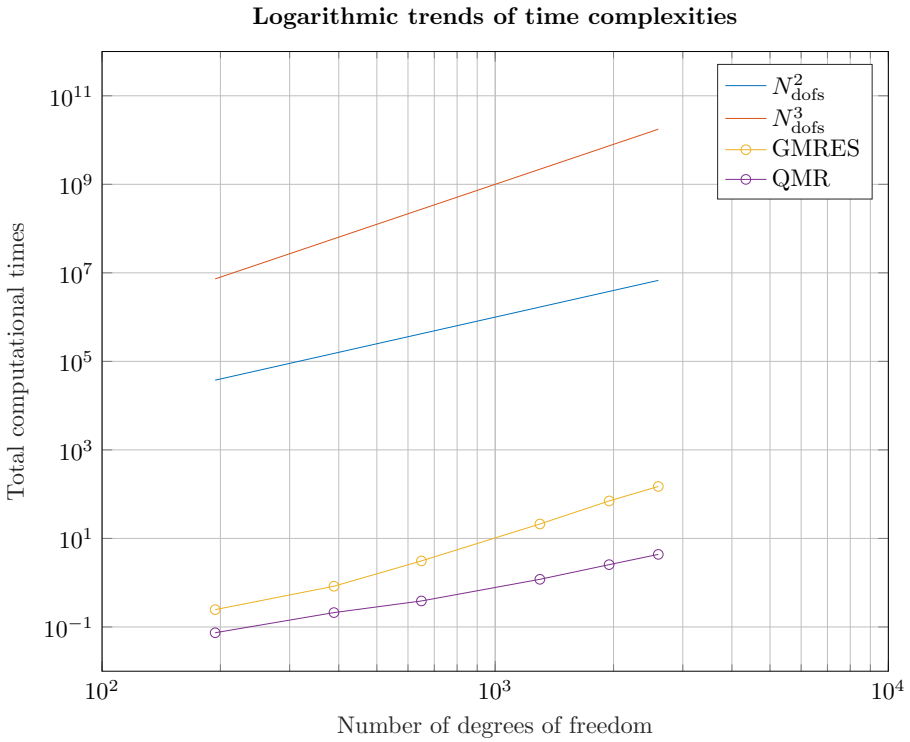


Table 4.2: Computational times for the GMRES (no restart) and QMR methods, truncated at the third decimal digit (left). Logarithmic plot of data (right). Times are measured in seconds.

Remark 4.8. Also in these cases the time complexity resemble the theoretical predictions, at least for sufficiently large values of N_{dofs}.

N_{dofs}	**CG**	**LU**
194	0.0874	0.033
389	0.149	0.047
649	1.495	0.074
1299	4.336	0.155
1949	9.457	0.192
2599	32.304	0.222

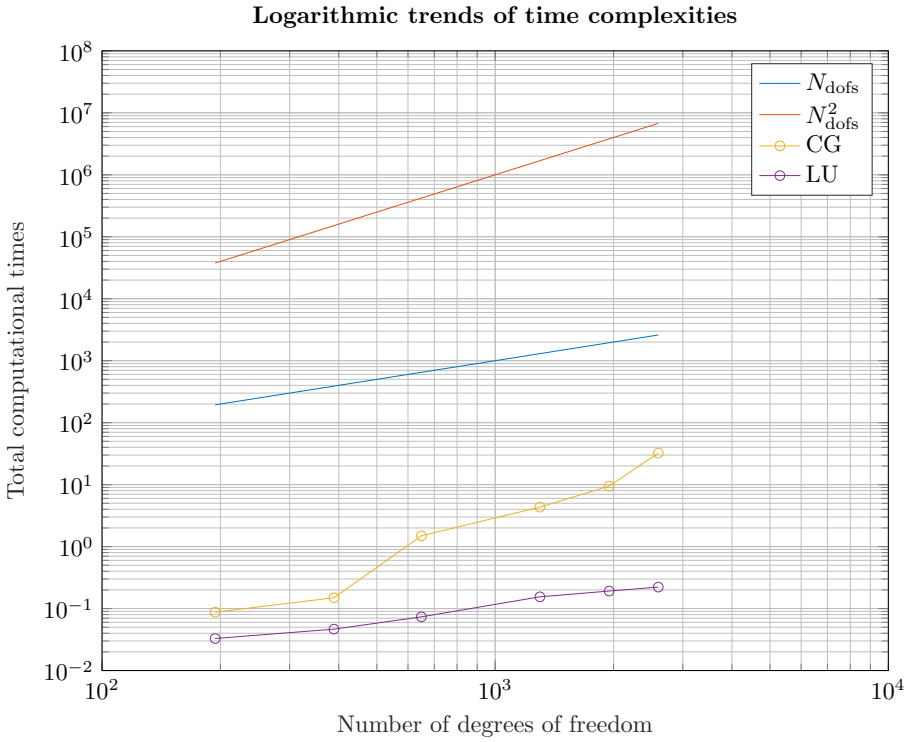


Table 4.3: Computational times for the CG and LU methods, truncated at the third decimal digit (left). Logarithmic plot of data (right). Times are measured in seconds.

Remark 4.9. In the case of the CG methods, the test cases seems to be insufficient to conclude an agreement with

the expected scaling. As for LU, the complexity seems to scale as $O(N_{\text{dofs}})$ whereas the theory predicts $O(N_{\text{dofs}}^3)$. This is possibly due to the fact that the matrices have a sparse structure, alongside with the optimization strategies employed by the SuperLU solver.

4.4 Mesh comparisons

The simulations shown so far are mainly done on graphs whose edges share the same length. In this Section we examine the issues related to using meshes with non uniform edge lengths.
In these cases we use two different approaches:

- **N-type mesh**: the simplest approach, uniformly dividing each edge in N subarcs without taking into account their possibly different length.
- **h-type mesh**: each edge is uniformly divided by fixing the length of the step h. Since it is not possible to know *a priori* if each edge length is divisible by h, we take $N = \text{nint}\,(\frac{L}{h})$, where L denotes the length of the edge ($\text{nint}\,(\cdot)$ is the function that returns the nearest integer), and divide each edge by their respective value N.

The comparison between these two types of meshes is drawn by solving an eigenvalue problem for the Laplacian operator using the QZ algorithm. Concerning eigenvalue trends, theoretical results predict that the spectrum is expected to present a parabolic growth towards infinity. This behaviour is modeled by a Weyl-type asymptotic formula:

$$N(\lambda;\Delta) = \pi^{-1}\,|\Gamma|\,\lambda^{\frac{1}{2}}(1+o(1))$$

This estimate is proven in literature for symmetric trees in [12]. Moreover, this topic will be widely treated in Chapter 5. We expect even other graphs to follow this behaviour, under suitable conditions, being one-dimensional manifolds, as shown in [8].
The numerical tests do not display any notable discrepancy when considering the smallest eigenvalues of the operators. Nonetheless, the largest eigenvalues present very different trends between the two approaches. It is known in literature that such eigenvalues are not handled very well by numerical methods in these types of problems ([7], Section 8.2).
Results in Figures 4.13, 4.14 and 4.15 show that:

- **N-type mesh**: the largest eigenvalues diverge very quickly in a not very regular way. This behaviour is too fast and in disagreement with the theory.
- **h-type mesh**: the largest eigenvalues have a smooth behaviour and the speed of their growth seems to agree with literature. However, the last ones present a concave trend which is in contradiction with the theory.

A possible explanation for this type of behaviour can be found by analyzing the relation between the eigencouples of the discretized Hamiltonian with the ones of the original differential operator. Indeed, the spectrum of the discretized Hamiltonian is expected to resemble the leftmost part of its continuous counterpart. Furthermore, as the eigenvalue index increases, eigenvectors tend to have an increasingly oscillatory behaviour, which can be caught only by sufficiently sharp meshes. Since, when the edge length is not uniform, the N-type mesh has different steps on different edges, eigenvectors whose support contains edges where the mesh is coarser may be "skipped" and lost when solving the algebraic problem. In spite of these ones, some other eigencouples are found, whose corresponding eigenvalues are larger. We suppose that these eigenpairs still have a counterpart in the continuous spectrum, and they are caught by the N-type mesh since their eigenvectors present oscillations mainly on the edges with a sharper mesh.
To confirm our conjecture, we show a number of simulations considering three different graphs:

- **Barabási-Albert type graph:** this graph is built by constructing its adjacency matrix Through the Barabási-Albert algorithm and then placing the vertices on the circle centered in $(0,0)$ with radius equal to 1. The first mesh is built with a fixed step $h = 0.1$.
- **Coarse tree graph:** this graph is built from the starting vertex $(0,0)$, then after each bifurcation, which is made with a $\frac{\pi}{4}$ angle, the length of the edge is $\frac{1}{n^2}$ where n denotes the bifurcation level minus one. The first mesh is built with a fixed step $h = 0.05$.
- **Sharp tree graph:** This graph has the same structure of the previous one but the first mesh is built starting from a fixed step $h = 0.005$.

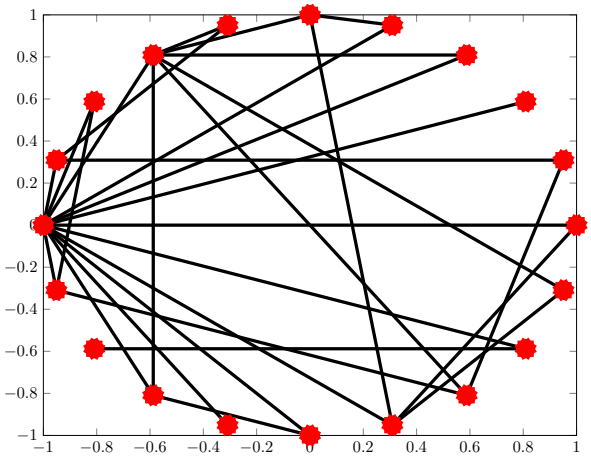


Figure 4.11: Planar Barabási-Albert graph.

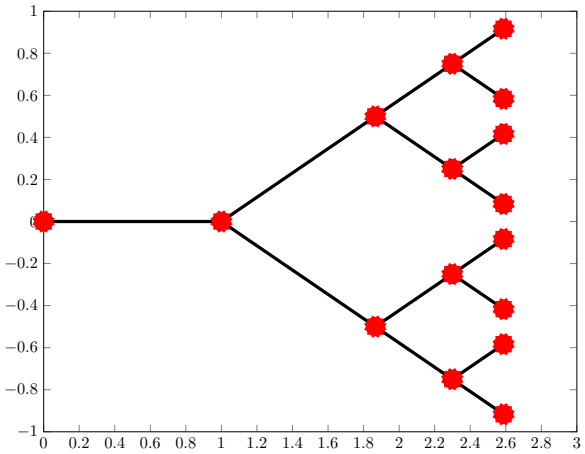


Figure 4.12: Tree graph. It start with the first two edges of length 1, afterwards each edge is divided by the number of bifurcations made before squared. We made the tests stopping at the fourth order of bifurcation.

Three different simulations are made on each graph. The first one is conducted with an h-type mesh. The second test is made with an N-type mesh, where N is chosen in such a way that the total number of degrees of freedom is comparable with the previous simulation (so that the number of eigencouples is almost the same). Finally, a third experiment is made considering a different h-type mesh, whose spatial step is equal to the one imposed on the shortest edge in the second simulation. This mesh is built in this way to test our "skipping" thesis; indeed, in the N-type mesh, we are getting some eigencouples whose eigenvectors are probably oscillating mainly in the sharper zones of the mesh, so we built an h-type mesh that is as sharp as these regions. Our goal is to find that the largest eigenvalues on both meshes present comparable values.
Results show that the largest eigenvalues diverge rapidly and atypically in the N-type mesh (green lines); whereas the h-type mesh cases present much smoother trends (red/blue lines). However they begin to show an inaccurate concave trend towards the last values (this is probably due to the discretization of the operator).
In addition, the simulations support the aforementioned "skipping" eigenvalue claim as we can see by comparing the blue lines with green ones in the plots.

Barabasi-Albert	**h-type mesh**	**N-type mesh (same dofs)**	**h-type mesh (minimum step)**
Step length	0.1	$\frac{376}{29} \cong 13$	$\frac{0.3129}{13} \cong 0.024$
Dofs	376	368	1629

Table 4.4: Barabasi-albert type graph mesh steps

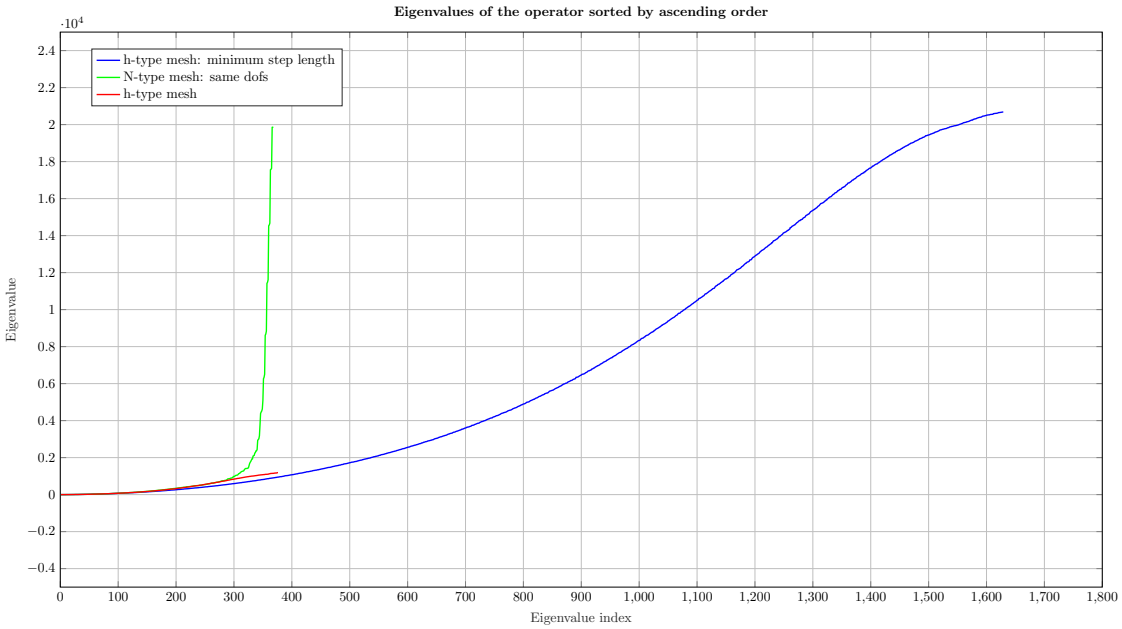


Figure 4.13: Eigenvalues for different mesh on Barabasi-Albert

COARSE tree	**h-type mesh**	**N-type mesh (same dofs)**	**h-type mesh (minimum step)**
Step length	0.05	$\frac{93}{15} \cong 6$	$\frac{\frac{0.1112}{6}}{0.0185} \cong$
Dofs	93	91	263

Table 4.5: Coarse tree graph mesh steps

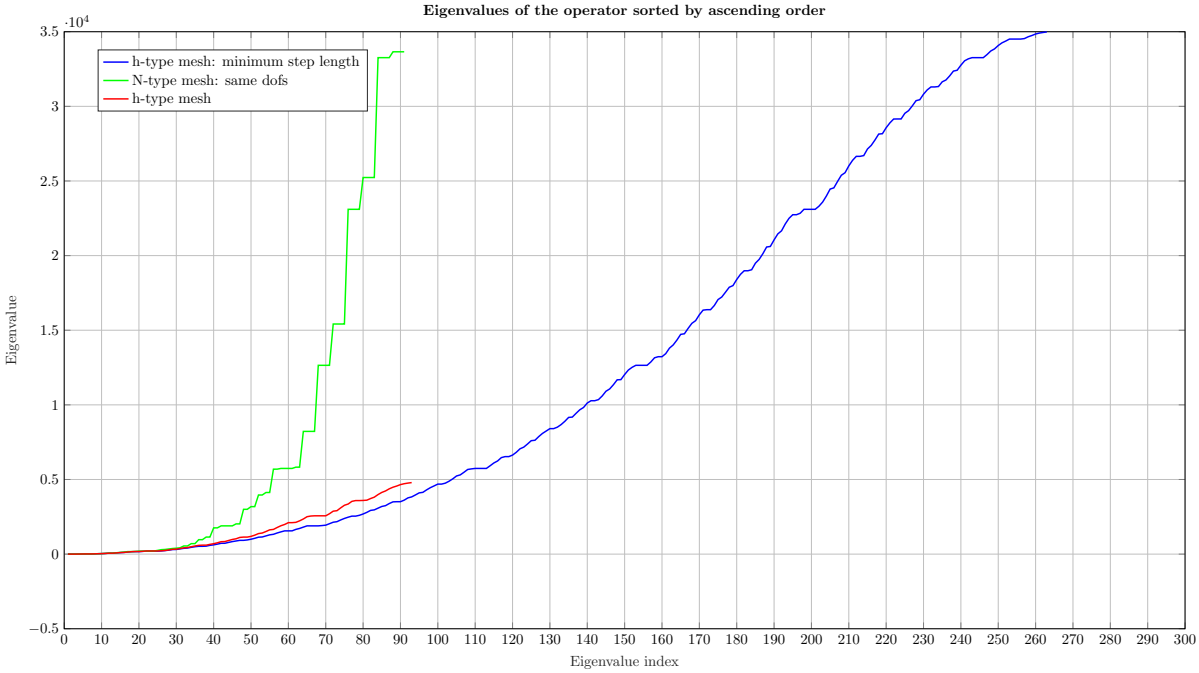


Figure 4.14: Eigenvalues for mesh comparison on "coarse" tree

SHARP tree	**h-type mesh**	**N-type mesh (same dofs)**	**h-type mesh (minimum step)**
Step length	0.005	$\frac{973}{15} \cong 65$	$\frac{0.1112}{65} \cong 0.0017$
Dofs	973	976	2849

Table 4.6: Sharp tree graph mesh steps

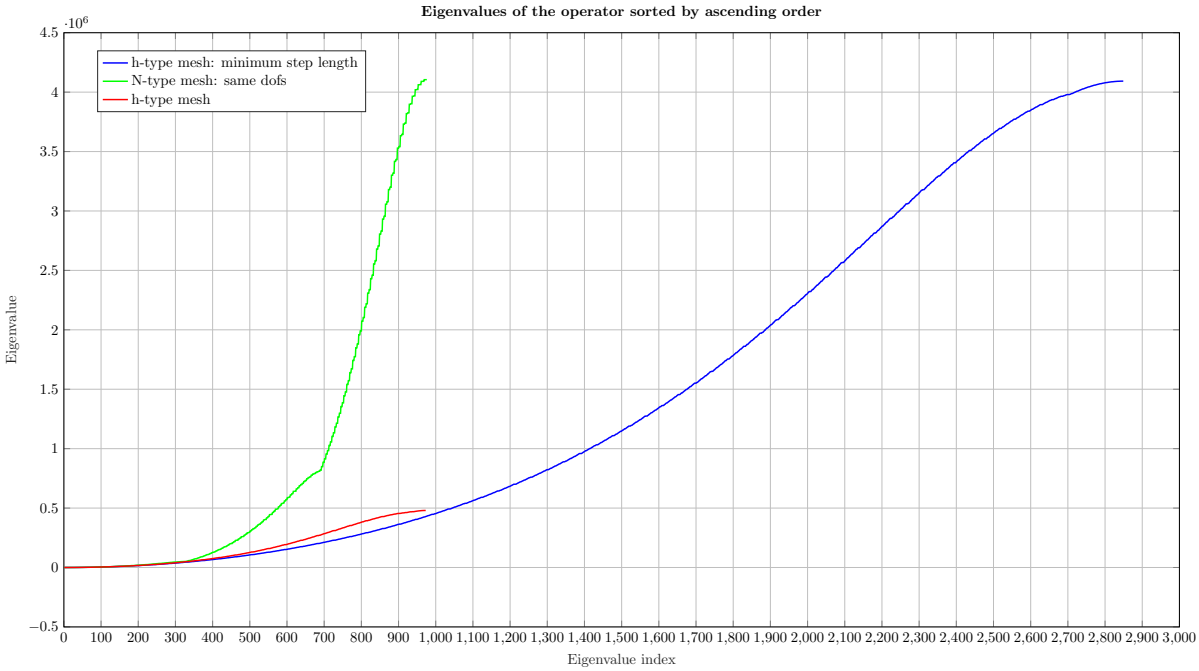


Figure 4.15: Eigenvalues for mesh comparison on "sharp" tree

Chapter 5

Elliptic Eigenvalue Problems on Metric Trees

5.1 Introduction on eigenvalue estimates for the Laplacian

In this Chapter we focus on studying metric tree graphs. These types of graphs are very important as they can be used as a first model approximation for a lot of applications and they are easy to study thanks to their symmetry. We present here a brief overview of the eigenvalue problem for the Laplacian on trees. A more detailed analysis can be found in [9], [10] and [12]. Our goal is to study the asymptotic behaviour of the eigenvalues, in particular to check the validity of the Weyl formula. We enounce here the classical version of this theorem ([13]):

Theorem 5.1 (Weyl, 1911)**.** *Let $N(\lambda, \Delta)$ be the Dirichlet eigenvalue counting function on a bounded domain Ω then*

$$N(\lambda, \Delta) \sim c_d \operatorname{vol}(\Omega) \lambda^{\frac{d}{2}}$$

where $c_d = \frac{w_d}{(2\pi)^d}$ is a constant depending only on the dimension d and w_d is the volume of the unit ball in $\mathbb{R}^d$, and $\operatorname{vol}(\Omega)$ *denotes the d-dimensional Lebesgue measure of Ω.*

Since we are working on graphs, which are one-dimensional manifolds, we aim to prove that Theorem 5.1 holds for them with $d = 1$. This result is stated in Theorem 5.8.

Remark 5.1 (Weyl formula in dimension $d = 1$)**.** The validity of the Weyl law in the one dimensional case can be easily checked. Indeed let, for simplicity, $\Omega = [0, a]$, then the Dirichlet eigenvalue problem becomes:

$$\begin{cases} \frac{\mathrm{d}^2}{\mathrm{d}x^2}\phi(x) = \lambda\phi(x), \\ \phi(0) = \phi(a) = 0. \end{cases}$$

Solving the boundary value problem, the Dirichlet eigenvectors are $\phi_k(x) = \sin\left(\frac{k\pi}{a}x\right)$ with corresponding Dirichlet eigenvalues $\lambda_k = \left(\frac{k\pi}{a}\right)^2$, for $k \in \mathbb{N}$. Let us consider now the eigenvalue counting function:

$$N(\lambda, \Delta) = \#\{k \in \mathbb{N} : \lambda_k < \lambda\} = \#\left\{k \in \mathbb{N} : \left(\frac{k\pi}{a}\right)^2 < \lambda\right\} = \max\left\{k \in \mathbb{N} : k < \frac{a\sqrt{\lambda}}{\pi} < \lambda\right\} \sim \frac{a}{\pi}\sqrt{\lambda}$$

Let us state a classical result for the characterization of the eigenvalues of an operator.

Theorem 5.2 (Min-max principle)**.** *We give here two version of the min-max principle:*

(i) Let A be a compact and self-adjoint operator on a Hilbert space H, and let λ_n be its eigenvalues sorted in increasing order and $\rho(u) = \frac{\|Au\|}{\|u\|}$ be its Rayleigh quotient.

$$\lambda_n = \inf_{X \in \Phi_n(H)} \sup_{u \in X} \rho(u)$$

Here $\Phi_n(H)$ is the space of n-dimensional sub-spaces of H.

(ii) Let A be a compact self-adjont operator on a Hilbert space H whose positive eigenvalues are listed in decreasing order and go to 0 as their index n goes to ∞. Then

$$\lambda_n = \min_{S_n} \max_{x \in S_n, \|x\|=1} \langle Ax, x \rangle$$

where S_n is a subspace of codimension n (which is the dimension of the quotient space $n = \dim(\frac{H}{S_n})$, equivalent to $\dim(H) - \dim(S_n)$ in the finite dimensional case) of the domain H of the eigenvalue problem.

Remark 5.2 (Neumann eigenvalues)**.** Using version (i) of THEOREM 5.2 we can prove that the eigenvalues μ_n of the Neumann problem for the Laplacian are less or equal than the ones of the corresponding Dirichlet problem λ_n. Indeed $H_0^1(\Omega) \subset H^1(\Omega)$, hence $\Phi_n(H_0^1(\Omega)) \subset \Phi_n(H^1(\Omega))$, thus the eigenvalues of the Neumann problem, exploiting the min-max principle, are given by

$$\mu_n = \inf_{X \in \Phi_n(H^1(\Omega))} \sup_{u \in X} \rho(u) \leq \inf_{X \in \Phi_n(H_0^1(\Omega))} \sup_{u \in X} \rho(u) = \lambda_n.$$

5.2 Tree graphs

5.2.1 Regular trees

Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a metric graph. For each edge $e \in \mathcal{E}$, let $|e|$ denote its length. Given $(x, y) \in \Gamma \times \Gamma$, the distance $\rho(x, y)$ between them is computed as the length of the minimum path connecting them. In this way, the metric topology on Γ is introduced in a natural way. When Γ is a rooted tree graph, with root vertex o, given $x \in \Gamma$, $\rho(x, o)$ is also denoted with $|x|$. It is possible to construct a partial ordering on the points of the tree graph. Indeed, given any two points $x, y \in \Gamma$ there exists a unique polygonal path $\langle x, y \rangle$ in Γ starting at x and ending at y. Thus we can write $x \prec y$ if $x \in \langle y, o \rangle$ and $x \neq y$. In the following, Γ denotes a possibly infinite metric tree graph.

Definition 5.1 (Generation)**.** The generation of a vertex v is:

$$\mathrm{gen}\,(v) := \#\{x \in \mathcal{V}(\Gamma) : x \prec y\}$$

Any edge e emanating from a vertex v ($e = \langle v, w \rangle$ with $v \prec w$) has $\mathrm{gen}\,(e) := \mathrm{gen}\,(v)$.

Definition 5.2 (Branching number)**.** The branching number $\mathrm{b}\,(v)$ of a vertex v is given by the number of branches emanating from v.

Remark 5.3. The only vertex with $\mathrm{gen}\,(v) = 0$ is $v = o$.

We assume that $\mathrm{gen}\,(v) < \infty$ for any $v \in \Gamma$ and $\mathrm{b}\,(v) > 1$ for $v \neq 0$. Let us define as $e_v^1, ..., e_v^{\mathrm{b}\,(v)}$ the edges emanating form $v \in \mathcal{V}$ and we denote as e_v^- the only edge which terminates at that vertex.

Definition 5.3 (Regular tree)**.** Γ is said to be regular if all the vertices with the same generation have equal branching numbers and all the edges with the same generation share the same length.

Remark 5.4 (Identification of a regular tree)**.** Using the definition of regular tree Γ, we can define two sequences that fully determine the tree:

$$\{b_n\} = \{b_n(\Gamma)\}, \quad \{t_n\} = \{t_n(\Gamma)\}, \qquad n = 0, 1, ...$$

such that

$$\mathrm{b}\,(v) = b_{\mathrm{gen}\,(v)}, \quad |v| = t_{\mathrm{gen}\,(v)}, \qquad \forall v \in \mathcal{V}$$

We have that $b_n \geq 2, \ \forall n > 0$ and $\{t_n\}$ is a strictly increasing sequence starting from 0.

Definition 5.4 (Height)**.** The quantity

$$h(\Gamma) := \lim_{n \to \infty} t_n = \sup_{x \in \Gamma} |x|$$

is called height of Γ.

Definition 5.5 (Branching function and reduced height)**.** We define the branching function of Γ as:

$$g_\Gamma(t) := \#\{x \in \Gamma : |x| = t\}, \quad 0 \le t < h(\Gamma)$$

Notice that

$$g_\Gamma(0) = 1; \qquad g_\Gamma(t) = b_0...b_n, \quad t_n < t \le t_{n+1}, \quad n = 0, 1, ...$$

The reduced height of Γ is defined as:

$$L(\Gamma) := \int_0^{h(\Gamma)} \frac{\mathrm{d}t}{g_\Gamma(t)}$$

We refer to Section 1.2 to define the natural measure, integration and Lebesgue spaces on Γ. It is easy to show from the Definition of branching function that the measure of Γ is equal to $\int_\Gamma g_\Gamma(t)\,\mathrm{d}t$.

5.2.2 Exploiting the symmetries: orthogonal decomposition of $L^2(\Gamma)$

In what follows, we assume that Γ is a regular tree and, for generality, that $\#\mathcal{E} = \#\mathcal{V} = \infty$. In particular, interesting results are shown to hold for tree graphs with finite measure, for which $|e|$ goes to 0 as $\mathrm{gen}\,(e)$ goes to ∞. Notice that trees with a finite number of branches have their generating sequence $\{t_n\}$ vanishing from a fixed index N_0. A graph is compact if and only if it has a finite number of edges, which is naturally the case for our simulations. Most results will be proven at first for compact trees and afterwards extended to more general cases. In order to produce a useful decomposition of $L^2(\Gamma)$, a particular class of subtrees is needed.

Definition 5.6 (Subtrees)**.** For any vertex $v \in \mathcal{V}$ and for any edge $e = \langle v, w\rangle, v \prec w$, we define the following subtrees:

$$T_v = \{x \in \Gamma : x \succeq y\}, \qquad T_e = e \cup T_w.$$

We can see that $T_o = \Gamma$ and due to the regularity of Γ, all subtrees T_e s.t. $\mathrm{gen}\,(e) = k$ are isomorphic to a tree Γ_k identified by the generating subsequences:

$$\begin{cases} b_0(\Gamma_k) = 1, & b_n(\Gamma_k) = b_{k+n}(\Gamma_k), \quad n \in \mathbb{N} \\ t_0(\Gamma_k) = 0, & b_n(\Gamma_k) = b_{k+n}(\Gamma_k), \quad n \in \mathbb{N} \end{cases} \tag{5.1}$$

It follows from Definition 5.5 that the branching function g_{Γ_k} is given by:

$$g_{\Gamma_k}(t) = \frac{g_\Gamma(t_k + t)}{b_0...b_K} = \frac{g_\Gamma(t_k + t)}{g_\Gamma(t_k+)}, \qquad k = 0, 1, ...$$

where $g(t_k+)$ denotes the right limit as t approaches t_k of $g_\Gamma(t)$. By Definition 5.6 it's true that:

$$T_v = \bigcup_{1 \le j \le \mathrm{b}\,(v)} T_{e_v^j} \qquad \forall v \in \mathcal{V}(\Gamma)$$

Therefore we can notice that any subtree T_v with $\mathrm{gen}\,(v) = k$, can be identified with the union of b_k copies of the tree Γ_k emanating from the common root v.

Definition 5.7 ($\mathcal{F}_T$ symmetric spaces)**.** Let $T \subset \Gamma$ be a subtree. The functional space:

$$\mathcal{F}_T = \{f \in L^2(\Gamma) : f(x) = 0,\ x \notin T \text{ and } f(x) = f(y),\ x, y \in T,\ |x| = |y|\}$$

$\mathcal{F}_T$ consists of all symmetric (i.e. depending only on $|x|$) functions from $L^2(\Gamma)$ with support fully contained in T.

Now, we consider the subtrees T_e and T_v given by Definition 5.6 and we denote by $\mathcal{F}_e = \mathcal{F}_{T_e}$ and $\mathcal{F}_v = \mathcal{F}_{T_v}$ their associated subspaces. The subspaces $\mathcal{F}_{e_v^j}, j = 1, ..., \mathrm{b}\,(v)$ generated by the edges emanated from v are mutually orthogonal by definition. We define $\tilde{\mathcal{F}}_v = \bigoplus_{1 \le j \le \mathrm{b}\,(v)} \mathcal{F}_{e_v^j}$, which is contained in $\mathcal{F}_v$ and we denote $\mathcal{F}'_v = \tilde{\mathcal{F}}_v \ominus \mathcal{F}_v$.

Theorem 5.3. *Let Γ be a regular metric tree with root vertex o and $\mathrm{b}(o) = 1$. Then, the subspaces $\mathcal{F}'_v$, $v \in \mathcal{V}(\Gamma)$ are mutually orthogonal and orthogonal to $\mathcal{F}_\Gamma$. Moreover*

$$L^2(\Gamma) = \mathcal{F}_\Gamma \oplus \left(\bigoplus_{v \in \mathcal{V}(\Gamma)} \mathcal{F}'_v \right)$$

and this decomposition reduces the Dirichlet Laplacian and the Neumann Laplacian on Γ.

The meaning of Theorem 5.3 is explained in the following Section.

5.3 Eigenvalue problems for the Laplacian operator on trees

5.3.1 Reduction of the Laplacian

Let us denote as Δ_k the Laplacian on each subtree Γ_k defined by 5.1. Let us denote their restrictions on symmetric subspaces given by DEFINITION 5.7 as $\mathcal{U}_0 = \Delta \restriction \mathcal{F}_\Gamma$ and $\mathcal{U}_k = \Delta_k \restriction \mathcal{F}_{\Gamma_k}$.

Theorem 5.4. *Let $v \in \mathcal{V}(\Gamma)$ and $\operatorname{gen}(v) = k > 0$. Then, the operator $\Delta \restriction \mathcal{F}'_v$ is unitary equivalent to the orthogonal sum of $(b_k - 1)$ copies of the operator $\mathcal{U}_k$.*

The main idea of the proof of THEOREM 5.4 is based on the fact that $\Delta \restriction \mathcal{F}_v$ is the orthogonal sum of b_k copies of the operator $\mathcal{U}_k$, and, since we defined $\mathcal{F}'_v = \tilde{\mathcal{F}}_v \ominus \mathcal{F}_v$, we can expect the withdrawal of one copy. This results are shown in [12] and in deeper detail in [10].
These restrictions are introduced to allow us to consider operators acting on functions depending on x only through $|x|$. Thus, it is possible to prove an equivalence between such operators and a new family of operators acting on functions defined on a segment. Naturally, these type of operators are much easier to study then $\mathcal{U}_0$ and $\mathcal{U}_k$. In order to do so, let $I_j = (t_{j-1}, t_j), j \in \mathbb{N}$. Let us consider a family of operators $\{\mathbf{A}_k\}$ such that $\operatorname{Dom}(\mathbf{A}_k)$ (which denotes the domain of the operator $\mathbf{A}_k$) consists of functions u on $[t_j, h(\Gamma))$, such that $u \restriction I_j \in H^2(I_j)$ for any $j > k$, i.e.

$$\sum_{j>k} \int_{I_j} \left(|u''|^2 + |u|^2\right) \mathrm{d}t < \infty,$$

and the following boundary condition at t_k and matching conditions at the point t_j, $j > k$ are satisfied:

$$\begin{gathered} u(t_k) = 0; \qquad u(t_j+) = b_j^{\frac{1}{2}} u(t_j-), \qquad j > k; \\ u(t_j+) = b_j^{-\frac{1}{2}} u(t_j-), \qquad j > k. \end{gathered} \tag{5.2}$$

The operators $\mathbf{A}_k$ act on this domain as

$$(\mathbf{A}_k u)(t) = -u''(t), \quad t \neq t_j,\ j \geq k$$

and they are self-adjoint and non-negative. To each operator $\mathbf{A}_k$ we associate a quadratic form

$$\mathbf{a}_k[u] = \sum_{j>k} \int_{I_j} |u'|^2 \ \mathrm{d}t, \quad u \in \operatorname{Quad}(\mathbf{A}_k) \tag{5.3}$$

whose domain is denoted by

$$\operatorname{Quad}(\mathbf{A}_k) = \{u \in L^2(t_k, h(\Gamma)) : u \restriction I_j \in H^1(I_j)\ \forall j > k,\ \sum_{j>k} \int_{I_j} |u'|^2 \ \mathrm{d}t < \infty \text{ and the conditions (5.2) are satisfied}\}.$$

Lemma 5.1. *$\mathcal{U}_k$ are unitary equivalent to $\mathbf{A}_k$ for any $k = 0, 1, ...$*

The proof of LEMMA 5.1 is given in [12]. From the previous Lemma, THEOREM 5.4 and THEOREM 5.3 it is possible to deduce:

Theorem 5.5. *Let Γ be the regular tree with the generating sequences $\{b_n\}$ (with $b_0 = 1$) and $\{t_n\}$. Then*

$$\Delta \sim \mathbf{A}_0 \oplus \left(\bigoplus_{k=1}^{\infty} \mathbf{A}_k^{[b_0 \ldots b_{k-1}(b_k - 1)]} \right)$$

where $\sim$ means unitary equivalent and $\mathbf{A}^{[r]}$ means the orthogonal sum of r copies of a self-adjoint operator $\mathbf{A}$.

By (5.3) we can see that $\operatorname{Quad}(\mathbf{A}_i) \subset \operatorname{Quad}(\mathbf{A}_j)$, $\forall i < j$ and that $\mathbf{a}_k = \mathbf{a}_0 \restriction \operatorname{Quad}(\mathbf{A}_k)$, $\forall k \in \mathbb{N}$. These relations imply that the spectral properties of all the operators $\mathbf{A}_k$ are determined only by the properties of the operator $\mathbf{A}_0$. The same applies, by THEOREM 5.5, to the operator Δ.

Theorem 5.6. *Let $\{\mathbf{A}_k\}$, $k = 0, 1, ...$ be the operators in $L^2(t_k, h(\Gamma))$ defined in this Section. Then:*

(i) If $\mathbf{A}_0$ is positive definte, then the same is true for any operator $\mathbf{A}_k$, $k \in \mathbb{N}$ and

$$\min \sigma(\mathbf{A}_0) < \min \sigma(\mathbf{A}_i) < \min \sigma(\mathbf{A}_j), \quad \forall i < j$$

where $\sigma(D)$ represents the spectrum of the operator D.

(ii) If the spectrum of $\mathbf{A}_0$ is discrete, then the same is true for any operator $\mathbf{A}_k$, $k \in \mathbb{N}$.

(iii) If the spectrum of $\mathbf{A}_0$ is discrete, then

$$\min \sigma(\mathbf{A}_k) \to \infty \ \ as \ \ k \to \infty$$

We can deduce by THEOREM 5.3 and THEOREM 5.5 that

$$\sigma_p(\Delta) = \bigcup_{k=0}^{\infty} \sigma_p(\mathbf{A}_k); \quad \sigma(\Delta) = \overline{\bigcup_{k=0}^{\infty} \sigma(\mathbf{A}_k)}$$

Using this results together with THEOREM 5.6 we can obtain:

Corollary 5.1. *The following hold true:*

(i) The Dirichlet Laplacian Δ on a regular tree is positive definite iff the operator $\mathbf{A}_0$ is positive definite. Moreover,

$$\inf \sigma(\Delta) = \inf \sigma(\mathbf{A}_0)$$

(ii) The spectrum of Δ is discrete iff the spectrum of $\mathbf{A}_0$ is discrete.

5.3.2 Weyl-type asymptotic formula for trees

As previously anticipated, the theory of metric trees can be used to build an estimate for the eigenvalues of the Laplacian. The asymptotics is proven at first for compact trees and can be extended afterwards to generic trees of finite measure. In order to prove the Weyl formula, we have first to exhibit the following Lemma.

Lemma 5.2. *Let Γ be a compact metric tree and $V \in L^{\infty}_{+}(\Gamma)$. Then, for any $n \in \mathbb{N}$ there exist points $x_j \in \Gamma$, $j = 1, ..., k$ such that $k \leq n$ and the inequality*

$$\int_{\Gamma} |u|^2 \, V \, \mathrm{d}x \leq \frac{|\Gamma| \int_{\Gamma} V \, \mathrm{d}x}{(n+1)^2} \, \|u'\|_2^2$$

holds for any function $u \in L^{1,2}(\Gamma)$ satisfying the conditions $u(x_j) = 0$, $\forall j$.

Here $u \in L^{1,2}(\Gamma)$ means that $\|u'\|_2 < \infty$. The full proof of LEMMA 5.2 can be found at [11]. The idea is to consider all the subtrees (DEFINITION 5.6) emanating from the root o, where the Dirichlet boundary condition is imposed. Using the equality $u(x) - u(o) = \int_0^{t_0} u'(t)dt$, where x is a generic point of the subtree and the function is integrated on the path $\langle x, o \rangle$, we can write for each of them the inequality $|u(x) - u(o)|^2 = t_0 \int_0^{t_0} |u'(t)|^2 \, \mathrm{d}t \leq \left| T_{e_j^o} \right| \int_{T_{e_j^o}} |u'(x)|^2 \, \mathrm{d}x$, $\forall j$. Then, it is possible to deduce a similar bound for the term $\int_{T_{e_j^o}} |u(x) - u(o)|^2 \, V \, \mathrm{d}x$. Using known results on super additive functions on trees (see [11], we do not go into details here), we get the inequality of the thesis by imposing the Dirichlet condition of the root o in $k \leq n$ points.
We now provide the main result on the asymptotic behaviour expected for the eigenvalues of the operator Δ.

Theorem 5.7. *The following hold true:*

(i) Let Γ be a regular tree and $h(\Gamma) < \infty$. Then the spectrum $\sigma(\Delta)$ is discrete.

(ii) Suppose in addition that $|\Gamma| < \infty$. Then the Weyl asymptotic formula for the eigenvalue counting function of Δ is satisfied:

$$N(\lambda; \Delta) = \pi^{-1} \, |\Gamma| \, \lambda^{\frac{1}{2}} (1 + o(1)).$$

Main ideas of the proof. The proof of this Theorem can be found in [12] and it exploits the results discussed in the previous pages. We briefly report the main concepts of the proof.
To prove *(i)* we can simply show that the spectrum of $\mathbf{A}_0$ is discrete, then the thesis follows by means of COROLLARY 5.1. This is done by computing the inverse of the Rayleigh quotient for the operator $\mathbf{A}_0$, which, using the definition of the operator, can be written as:

$$\frac{\int_0^{h(\Gamma)} |\varphi(t)|^2 \, g_\Gamma(t) \, \mathrm{d}t}{\int_0^{h(\Gamma)} |\varphi'(t)|^2 \, g_\Gamma(t) \, \mathrm{d}t}, \quad \varphi \in H^1_{loc}((0, h(\Gamma)), g_\Gamma), \;\; \varphi(0) = 0$$

Then, substituting in the integration the reduced height as if it were the arc parameter, $s = s(t) = \int_0^t \frac{d\tau}{g_\Gamma(\tau)}$, we obtain:

$$\frac{\int_0^{L(\Gamma)} |\psi(s)|^2 \, W(s) \, \mathrm{d}s}{\int_0^{L(\Gamma)} |\psi'(s)|^2 \, \mathrm{d}s}, \quad \psi \in H^1_{loc}(0, h(\Gamma)), \;\; \psi(0) = 0. \tag{5.4}$$

where $W(s)$ is such that $\int_0^{L(\Gamma)} \sqrt{W(s)} \, \mathrm{d}s = \int_0^{L(\Gamma)} g_\Gamma(t(s)) \, \mathrm{d}s = h(\Gamma) < \infty$. This bound can be used to ensure the compactness of the operator generated by the Rayleigh quotient. Consequently, its spectrum is discrete, and its eigenvalues respect the property $\lambda_n < \left(\frac{Ch(\Gamma)}{n}\right)^2$, thus clustering towards 0 as $n \to \infty$.
Being 5.4 the inverse of the Rayleigh quotient of $\mathbf{A}_0$, we know that even its spectrum is discrete and we have the bound $\min(\sigma(\mathbf{A}_0)) \geq (Ch(\Gamma))^{-2}$. We can prove in an analogous way the same thing for $\mathbf{A}_k$ with an increasing bound for $\min(\sigma(\mathbf{A}_k))$ which gives us Proposition (iii) of THEOREM 5.6. The discreteness of the spectrum of Δ is therefore proven.
To prove point *(ii)* we have to consider again the inverse of the Rayleigh quotient of Δ

$$\frac{\int_\Gamma |f|^2 \, \mathrm{d}x}{\int_\Gamma |f'|^2 \, \mathrm{d}x}, \quad f(o) = 0.$$

Then, given a real number $\varepsilon > 0$, exploiting the fact that $|\Gamma|$ is finite, we can find a compact subtree $T_\varepsilon \subset \Gamma$ such that $|\Gamma \setminus t_\varepsilon| < \varepsilon$. Let us consider the operator $\mathbf{T}_{\Gamma,V}$ related to the following Rayleigh quotient

$$\frac{\int_\Gamma V \, |f|^2 \, \mathrm{d}x}{\int_\Gamma |f'|^2 \, \mathrm{d}x}, \quad f(o) = 0.$$

and let V_ε be the characteristic function of the aforementioned tree. We can write

$$\Delta^{-1} = \mathbf{T}_{\Gamma,1} = \mathbf{T}_{\Gamma,V_\varepsilon} + \mathbf{T}_{\Gamma,1-V_\varepsilon} \tag{5.5}$$

Now, consider the inequality of LEMMA 5.2 with $V = V_\varepsilon$. Dividing both members by $\|u\|_2^2$, we obtain the estimate for the Rayleigh quotient $\rho(\mathbf{T}_{\Gamma,V_\varepsilon}) \leq |\Gamma| \, n^{-2} \int_G V_\varepsilon \, \mathrm{d}x$ on a space of codimension $k \leq n$ (which is the number of fixed Dirichlet conditions of the Lemma). Using version (ii) of THEOREM 5.2 we can clearly see that

$$\lambda_n(\mathbf{T}_{\Gamma,V_\varepsilon}) \leq |\Gamma| \, n^{-2} \, |T_\varepsilon| \, .$$

This can be used to prove the asymptotic Weyl formula as shown in [11]

$$\pi \lambda^{\frac{1}{2}} \# \{n : \lambda_n(\mathbf{T}_{\Gamma,V_\varepsilon}) > \lambda\} = |T_\varepsilon| + o(1).$$

An analogous estimate for the remainder of the graph can be achieved (this result is proven in [11] too), which is:

$$\lambda_n(\mathbf{T}_{\Gamma,1-V_\varepsilon}) \leq |\Gamma| \, n^{-2} \varepsilon.$$

This inequality allows us to take the limit for $\varepsilon \to 0$ in (5.5), which leads to our thesis. □

5.4 Eigenvalue asymptotic estimates for generic graphs

In the previous Sections we have used the topological features of trees to prove an estimate for the eigenvalues of the Laplacian. We want to extend it to all graphs with finite measure. In order to do so, let us introduce a common procedure to reduce the problem on a general graph to a problem on a tree, technique usually called cutting cycles.

Proposition 5.1. Let Γ be a compact metric graph. Then there exist a compact metric tree T and a continuous mapping $\tau : T \to \Gamma$ such that the operator $\tau^* : u(x) \to u(\tau(x))$ defines an isometry for functions in $L^{1,2}(\Gamma)$ onto a subspace of finite codimension in $L^{1,2}(T)$.

Proof. The idea is to eliminate all loops from the compact connected graph Γ. In order to do so, we consider an edge of a loop $e_0 = \langle v, w\rangle$ and we build the graph $\Gamma_{e_0} = \Gamma \setminus \mathrm{Int}(e_0)$ such that $\mathcal{V}(\Gamma_{e_0}) = \mathcal{V}(\Gamma)$ and $\mathcal{E}(\Gamma_{e_0}) = \mathcal{E}(\Gamma) \setminus e_0$, where $Int(e)$ indicates the set of internal points of an edge e.
We now choose a random point $x \in \mathrm{Int}(e_0)$ and replace it with two new vertices $\{x_1, x_2\}$. Therefore, we can build the graph Γ_1 such that $\mathcal{V}(\Gamma_1) = \mathcal{V}(\Gamma) \cup \{x_1, x_2\}$ and $\mathcal{E}(\Gamma_1) = \mathcal{E}(\Gamma_{e_0}) \cup \{\langle v, x_1\rangle, \langle w, x_2\rangle\}$. Clearly, if we take in a natural way lengths from the graph Γ, we will have $|\Gamma_1| = |\Gamma|$. Define as $\tau_1 : \Gamma_1 \to \Gamma$ the map that is the identity in Γ_{e_0} and isometrically sends $\langle v, x_1\rangle$ into $\langle v, x\rangle$, $\langle w, x_2\rangle$ into $\langle w, x\rangle$. The function τ_1 is continuous and the corresponding map $\tau_1^* : u(x) \to u(\tau_1(x))$ is an isometry of the space $L^{1,2}(\Gamma)$ into $L^{1,2}(\Gamma_1)$. However, its image is made of all the functions such that $u(x_1) = u(x_2)$ and therefore its codimension is one (loss of one degree of freedom).
With this procedure we have built a mapping to a graph Γ_1 with at least one loop less than the original graph Γ. Being Γ compact, it always has a finite number of loops, thus we can construct a finite sequence of n graphs $\{\Gamma_j\}$ with maps $\tau_j : \Gamma_j \to \Gamma_{j-1}$ built with the procedure mentioned before, such that Γ_n has no loops. In this way, we have found the tree $T = \Gamma_n$ and the mapping $\tau = \tau_1 \circ \tau_2 \circ ...\tau_n$ that respect the definition in our thesis. □

Using these techniques and the Propositions about metric trees of Subsection 5.3.2 we can prove the following result.

Theorem 5.8 (Weyl formula for graphs)**.** *Let Γ be a connected graph of finite measure $|\Gamma|$, consider on it the eigenvalue problem for the Laplacian problem with homogeneous Dirichlet conditions imposed in at least one vertex x_0. The eigenvalues of the Laplacian operator follow the estimate:*

$$\#\{n : \lambda_n(\Delta) < \lambda\} = \pi^{-1}\lambda^{\frac{1}{2}}(|\Gamma| + o(1)).$$

We won't discuss the full proof of Theorem 5.8 here, however it can be found in [11]. The idea is to use Proposition 5.1 to prove a version of Lemma 5.2 for generic finite measure graphs. From that result we can exploit the same procedure used in the proof of point (ii) of Theorem 5.7 for the operator $\mathbf{T}_{\Gamma,V_\varepsilon}$.

5.5 Numerical confirmation of the Weyl asymptotic formula

In this Section we want to test via numerical simulations the validity of the Weyl asymptotic formula.

5.5.1 Handling numerical errors for large eigenvalues

Before going into the details we have to make few considerations about the numerical errors on the largest eigenvalues.
We already know (as was written in Section 4.4 and in [7]) that numerical methods used for eigenvalue problems on quantum graphs do not handle very well the largest eigenvalues. Therefore we want to decide a criterion to exclude the ones we think are less precise. We already made some considerations regarding the fact that largest eigenvalues in the N-type mesh diverge, whereas in the h-type mesh, while much more smoother, they present an atypical concave trend. Because of our "skipping conjecture" and the fact that the trend is smoother, we decide to perfrom tests on h-type meshes which are closer to uniform too.
In order to address the inaccurate concave trend, eigenvalues have been trimmed, so to get rid of those we think are not reliable. To do so, we decide to test the same problem on different meshes, gradually more refined, and then to compare each mesh with its successor. We consider trustworthy the eigenvalues that differ only less than 10% by their more refined counterpart. When we analyze our tests, we save the percentage for each mesh at which the eigenvalues start not to respect this condition; then we check that it remains almost the same through all tests and at the end we take the minimum one to trim the eigenvalues of the most refined mesh. This procedure completely excludes the eigenvalues that show the concave trend.
Using this technique to treat the largest eigenvalues of the operator Δ, we are ready to proceed and analyze their behaviour.

5.5.2 Weyl formula on compact trees

The first simulations to check the validity of the Weyl formula are made on trees. Instead of analyzing the counting function for the eigenvalues we study the inverse function $\lambda(N;\Delta)$ which gives the value of the eigenvalue with respect to their index N listed in ascending order. The inverse of the Weyl asymptotic formula (THEOREM 5.7) is:

$$\lambda(N;\Delta) = \left(\frac{\pi}{|\Gamma|}\right)^2 \cdot N^2(1+o(1))$$

Thus, we want to check that the eigenvalues increase with respect to their index with a quadratic behaviour with coefficient:

$$\left(\frac{\pi}{|\Gamma|}\right)^2.$$

To check the validity of the formula and in particular the truthiness of the coefficient, we show data coming from simulations on four different graphs:

- "$\frac{1}{n^2}$" Tree graph shown in FIGURE 5.1.
- "Power 3" Tree graph shown in FIGURE 5.2.
- "$\frac{1}{n^2}$" Tree graph rescaled by a factor 1.5.
- "Power 3" Tree graph rescaled by a factor 1.5.

This rescaling is important since the coefficient depends on $|\Gamma|$, which is the (one dimensional Lebesgue) measure of the graph Γ, i.e. the sum of all their edge lengths.

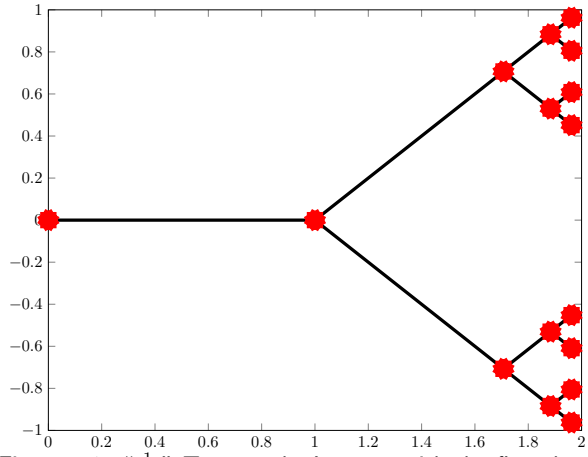


Figure 5.1: "$\frac{1}{n^2}$" Tree graph. It starts with the first three edges of length 1, afterward each edge is divided by the number of bifurcations made before squared. In this way its height is converging to $\frac{\pi^2}{6}+1$. We made the tests stopping at the fourth order of bifurcation.

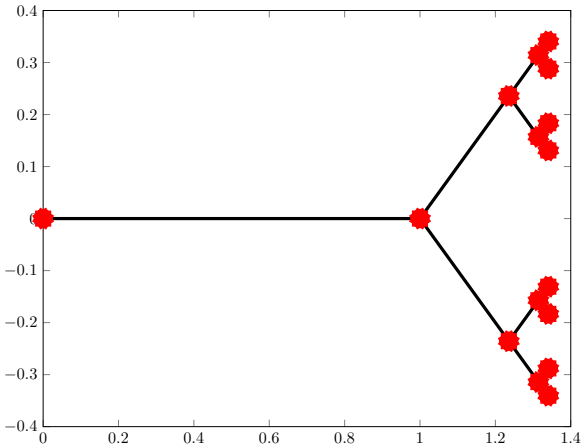


Figure 5.2: "Power 3" Tree graph. It starts with the first edge of length 1 and afterwards the length of the edges become one third of the previous one after each bifurcation. In this way its height is converging to $\frac{3}{2}$. We made the tests stopping at the fourth order of bifurcation.

If extended indefinitely, these two types of graphs have both converging height, however their measure is not. Indeed only the one in FIGURE 5.2 has its measure $|\Gamma|$ converging to 3, the measure of the one in FIGURE 5.1 diverges. Anyway, in this Section we are naturally studying compact tree graphs, stopping at the fourth order of bifurcation. We start from a fixed step $h = 0.016$ and we halve it progressively to make four simulations and trim the results using the method described in SUBSECTION 5.5.1. We obtain the following degrees of freedom:

Dofs of "$\frac{1}{n^2}$" Tree graph	295	604	1215	2437
Dofs of Re-scaled"$\frac{1}{n^2}$" Tree graph	452	906	1824	3661
Dofs of "Power 3" Tree graph	143	292	597	1197
Dofs of Re-scaled "Power 3" Tree graph	220	440	892	1797

Henceforth we will show only the plots referring to the "Power 3" Tree graph rescaled by the factor 1.5. This is because all the plots confirm the same trends, however all the calculated values are shown in some tables to check the validity of our statements.

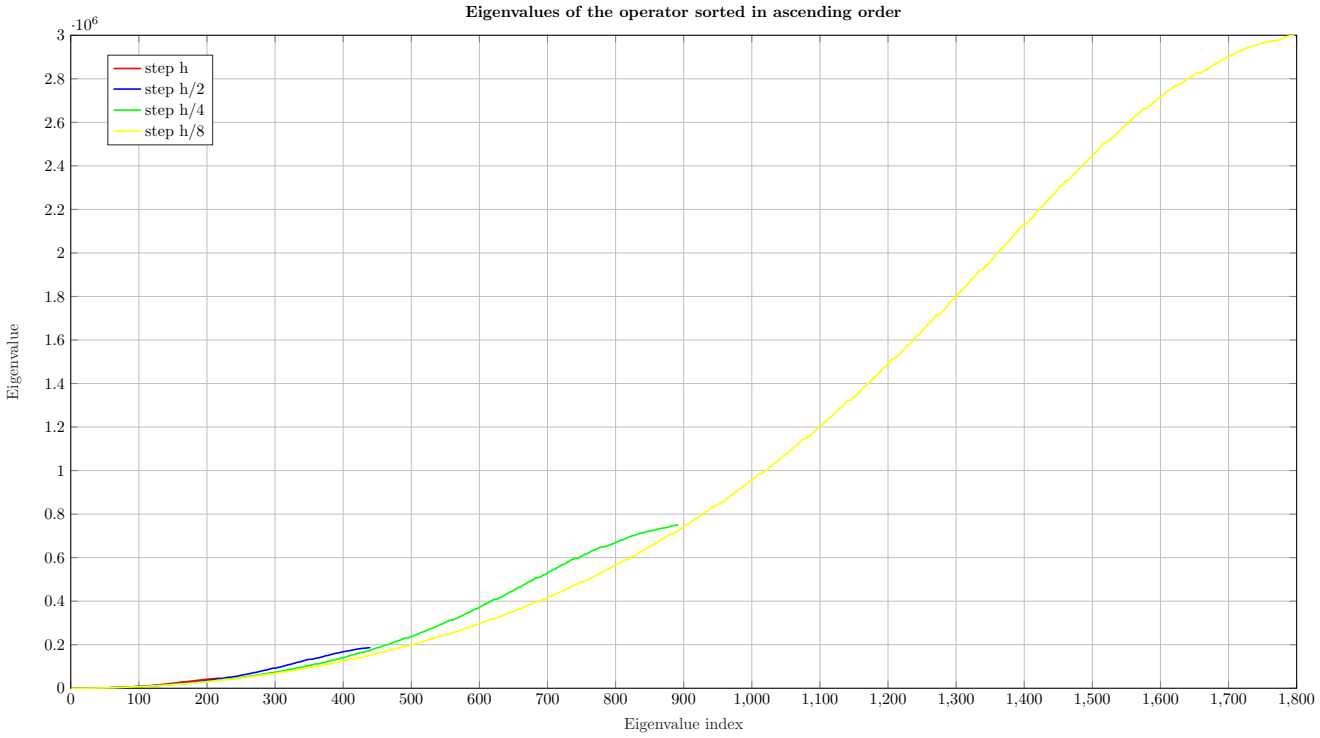


Figure 5.3: Eigenvalues for the rescaled "Power 3" Tree graph.

The fact that the trend is quadratic can be easily checked with a logarithmic plot.
Afterwards we compute, with a quadratic regression, the best second order polynomial to fit the data. We use the first coefficient found with this process as our estimated coefficient for the Weyl formula.

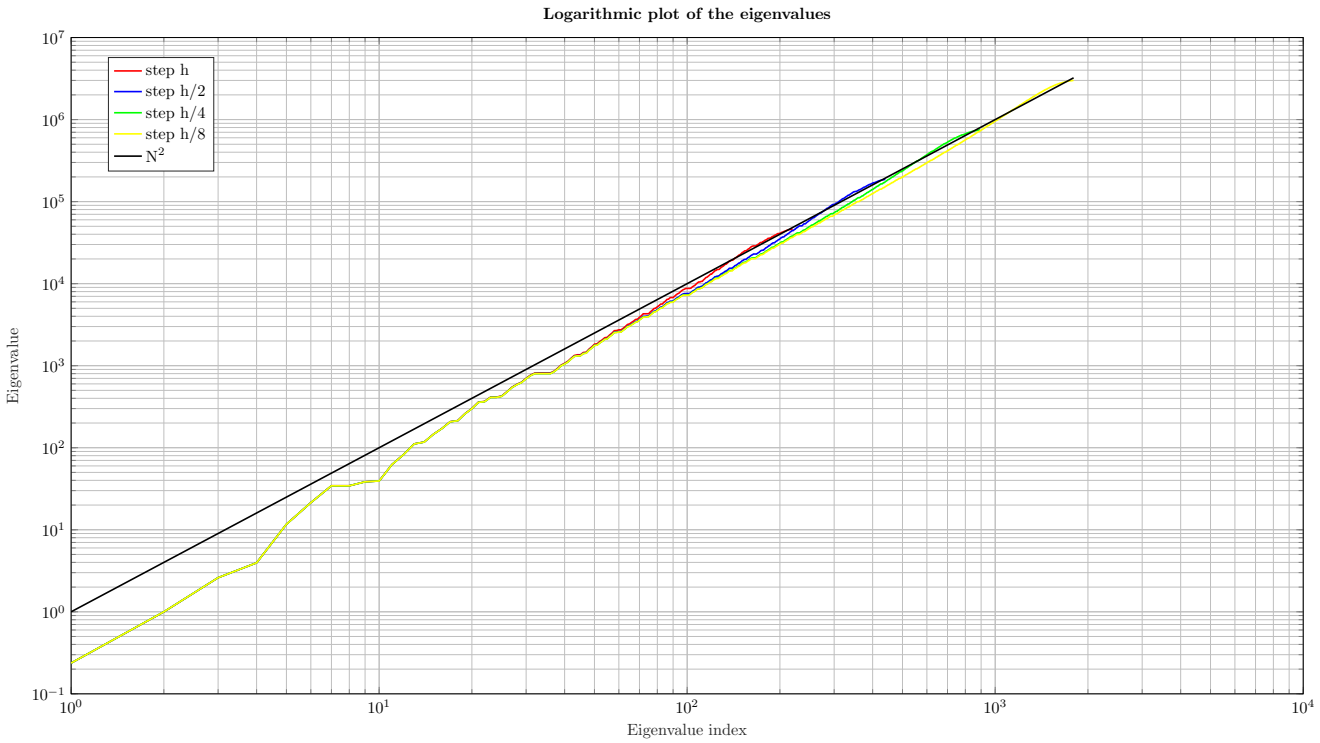


Figure 5.4: Logarithmic plot for the eigenvalues of the rescaled "Power 3" Tree graph.

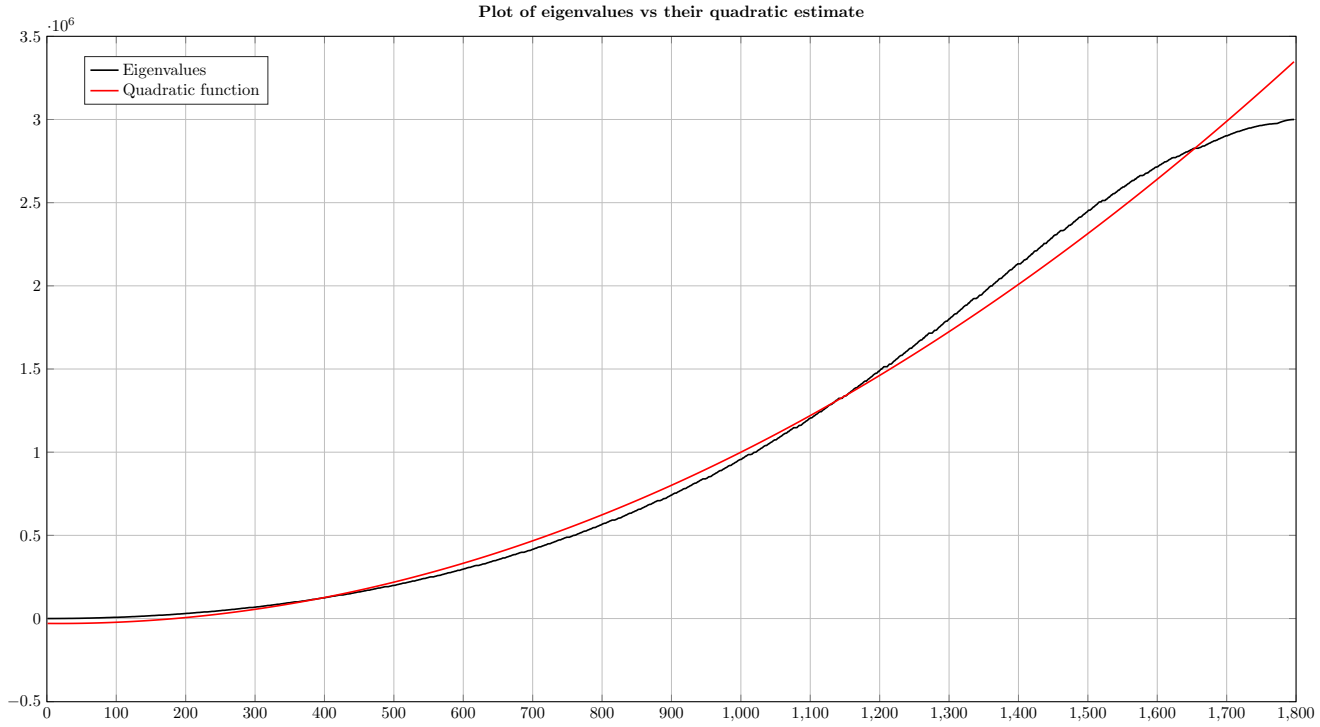


Figure 5.5: Quadratic estimate on the whole eigenvalues of the rescaled "Power 3" Tree graph.

After watching the raw data, we start using the trimming method described in SUBSECTION 5.5.1. For the first 3 tests we had the following percentage of trustworthy eigenvalues:

Percentage of "correct" eigenvalues of the "$\frac{1}{n^2}$" Tree graph	0.3763	0.4023	0.4165
Percentage of "correct" eigenvalues of the Re-Scaled "$\frac{1}{n^2}$" Tree graph	0.4226	0.4139	0.4189
Percentage of "correct" eigenvalues of the "Power 3" Tree graph	0.4169	0.4247	0.4255
Percentage of "correct" eigenvalues of the Re-Scaled "Power 3" Tree graph	0.4318	0.3727	0.4025

This leads us to the following trimmed plots:

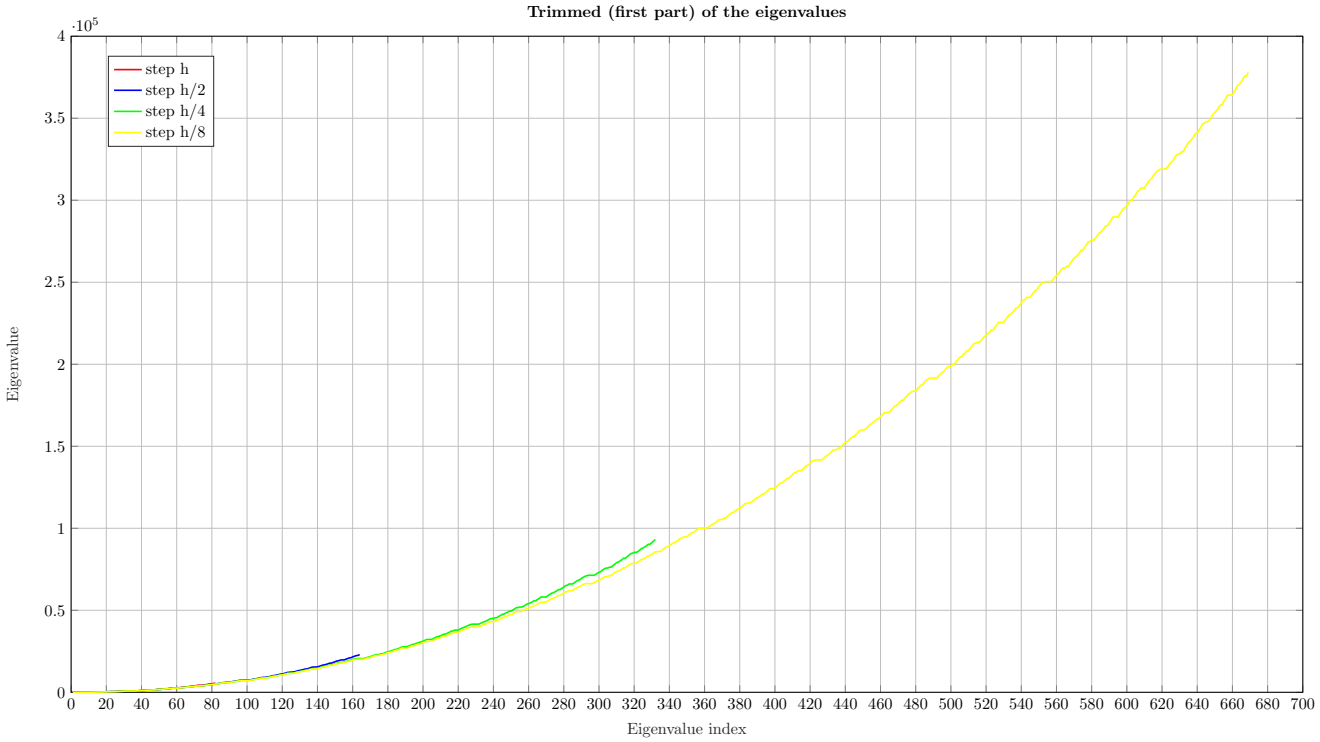


Figure 5.6: "Trimmed" eigenvalues of the Re-scaled "Power 3" Tree graph.

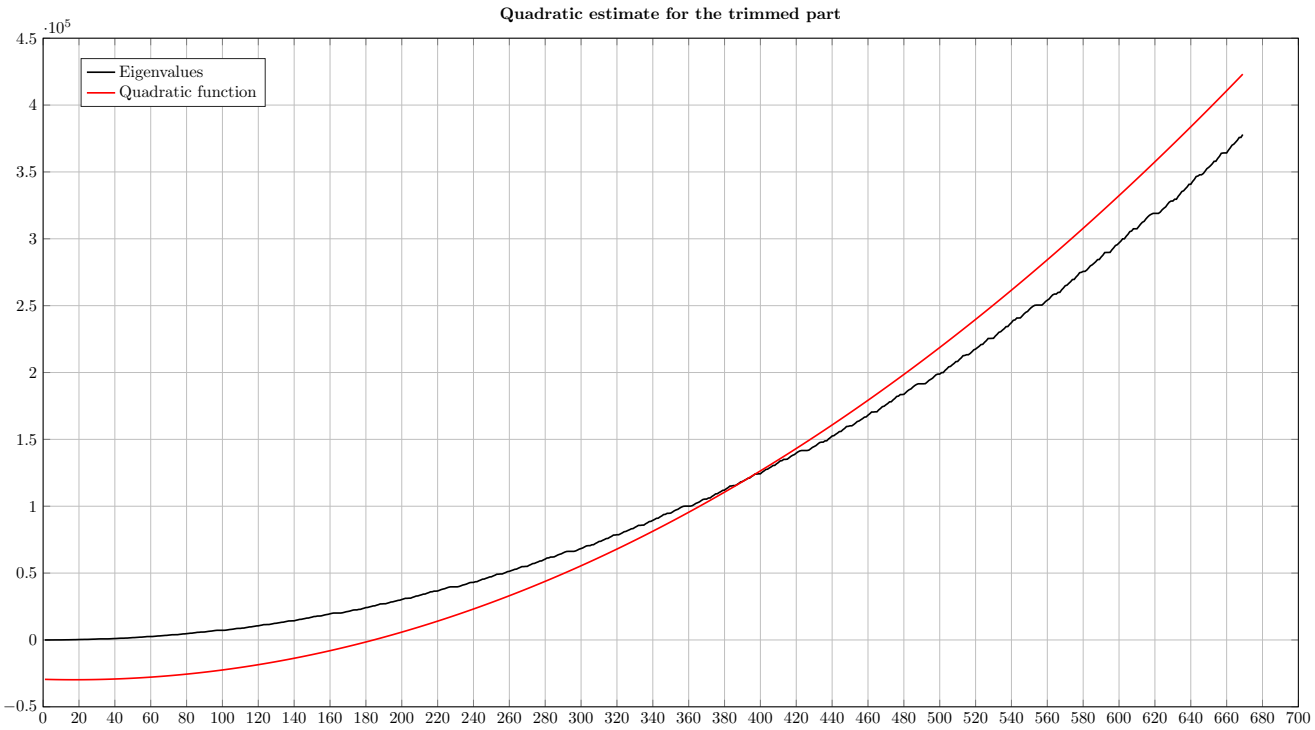


Figure 5.7: Quadratic estimate on the "trimmed" eigenvalues of the Re-scaled "Power 3" Tree graph.

Now, we can analyze the coefficient which we expect to be equal to $\left(\frac{\pi}{|\Gamma|}\right)^2$. Knowing that, we want it to be equal to $\left(\frac{\pi}{(\frac{8}{27}+\frac{4}{9}+\frac{2}{3}+1)\cdot 1.5}\right)^2 = 0.7569...$ by taking the actual measure of the graph.
These results are not totally true for our simulations, but as we will see the ratio between the expected coefficient and the one we find is the same even if we take different trees. Indeed the formula for our coefficient seems to remain the same but multiplied by approximately 1.2, factor which remains the same through all simulations and is not that bad of an error.
Moreover at least our results agree with the trend shown in the formula as, if we multiply the measure $|\Gamma|$ for a constant k, the coefficient will be divided by k^2.

	Whole eigenvalues coefficient	"Trimmed" eigenvalues coefficient	Real length coefficient estimate	Ratio between estimate and simulation result
"$\frac{1}{n^2}$" Tree graph	0.5805	0.4981	0.4129	1.2063
"Power 3" Tree graph	2.3989	2.1420	1.7029	1.2578
Re-Scaled "$\frac{1}{n^2}$" Tree graph	0.2578	0.2298	0.1835	1.2523
Re-scaled "Power 3" Tree graph	1.0661	0.9097	0.7568	1.2020

We can even check that the formula for the coefficient is of the right order w.r.t. $|\Gamma|$.
Indeed calculating the ratio between the original and the re-scaled versions of the graphs we obtain:

$$\frac{0.2298}{0.4981} = 0.4613 \cong 0.4444 = \left(\frac{1}{1.5}\right)^2$$

$$\frac{0.9097}{2.1420} = 0.4246 \cong 0.4444 = \left(\frac{1}{1.5}\right)^2$$

Even comparing the results between the two types of tree graphs confirm the formula for the coefficient, in fact:

$$\frac{2.1420}{0.4981} = 4.3003 \cong 4.1240 = \left(\frac{\frac{8}{9}+1+2+1}{\frac{8}{27}+\frac{4}{9}+\frac{2}{3}+1}\right)^2$$

5.5.3 Weyl formula on generic graphs

In Section 5.4 it was proven that Weyl asymptotic formula holds on a generic graph too. We show here the numerical confirmation of this fact by repeating the same tests of the previous Subsection on a Graphene-like graph (see Figure 4.6) which presents two loops. In the following table we can read the parameters used for the meshes and how much of the output is considered:

Dofs	295	604	1215	2437
Percentage of "valid" eigenvalues	0.4208	0.4216	0.4210	–

Everything is in agreement with the theory as it is shown in all the plots:

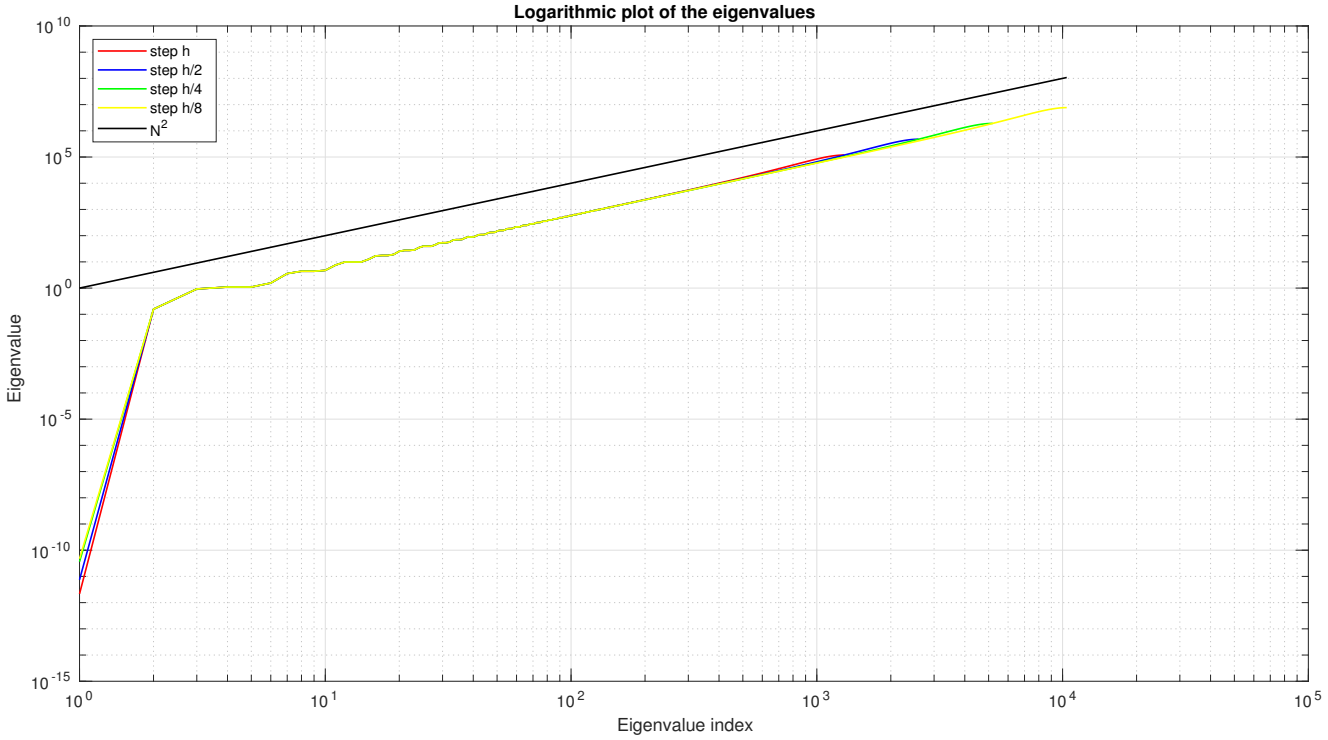


Figure 5.8: Trimmed plot for the eigenvalues vs. quadratic estimate.

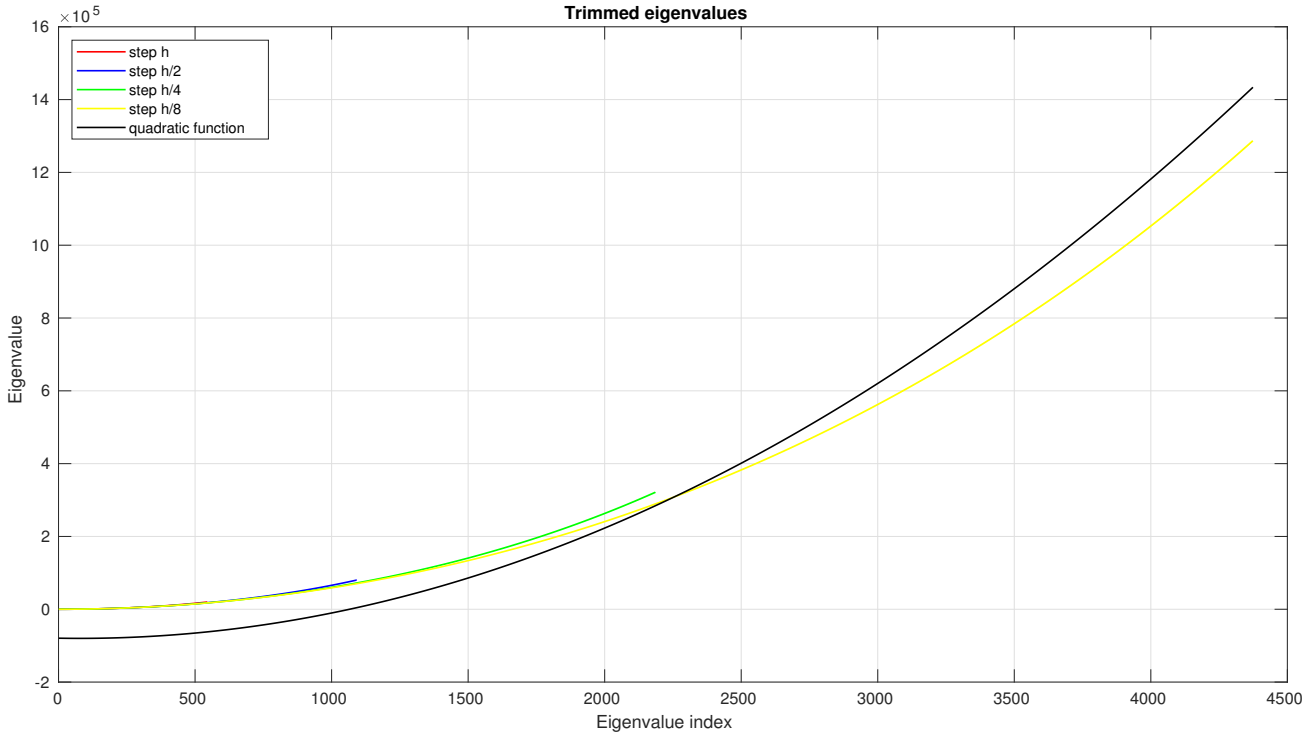


Figure 5.9: Logarithmic plot for the eigenvalues.

The computed coefficient, 0.0737 is about 1.2 times higher than the expected one as in the previous simulations indeed

$$1.2619 = \frac{0.0737}{(\pi/13)^2}.$$

We don't have a thesis for the reason why this ratio is maintained. The discretization technique may be a possible cause, but testing analogous cases on N-type meshes produce the same results. Therefore we don't have any idea on possible simulations that can be run to have a better understanding on what is generating this deviation from theoretical results.
Several other tests on different topologies and even with different boundary conditions were run; we don't feel the need to exhibit all these results here as no particular discrepancy from the previous ones and form the theory is shown.

Chapter 6

Conclusions and Future Work

This work aims to give the reader an introductory insight on the study of elliptic problems on graphs. A full presentation of the theoretical background is provided in order to define and study these kinds of problems from an analytical point of view. Afterwards, we present the tools to treat these models from a numerical standpoint and a C++ library is implemented through these techniques. The code can be found at

`https://github.com/sabba11/PACS_FEMG_Abbate_Di_Primio`.

and will be analyzed in Chapter 7. We think that our work can be very useful to study physical problems related to geometries which can be seen as metric graphs (i.e. one-dimensional manifolds in $\mathbb{R}^3$). The code has been tested and it shows good outputs for every case that has been inspected. Some other studies are conducted to check the convergence rates of all the implemented algorithms and to check the differences between the two types of meshes.
After the generic examination of differential operators on graphs, the investigation moves to eigenvalue problems on tree graphs. We think that this type of graph is very important for different reasons. First of all it can be used as a first approximation in several applications (for example fluid dynamics), as any geometry that can be represented as a graph with no loops is, indeed, a tree. Moreover, a lot of important results can be proven much more easily on trees, exploiting symmetries and absence of loops. These features allow to formulate in a simpler way problems as they were in one spatial dimension. The map displayed in Proposition 5.1 can then be used to generalize those results on all types of graphs.
Using this approach, the asymptotics of Theorem 5.8 is proven and it has been widely tested on different types of graphs. The formula is confirmed by our simulations, however a coefficient approximately 1.2 times larger is found, even though it changes as expected with respect to the measure of the graph. This main result shows that differential operators on graphs do have the behaviour of one dimensional operators, although defined on a geometry embedded in two or three dimensional spaces (the Theorem is in fact the generic Weyl Formula for dimension one). Further confirmation of this fact can be seen in Subsection 4.2.4 where changing the angle of bifurcation for trees does not affect in any way the eigenvalues of elliptic operators on them. This fact corroborates the idea that only the one dimensional structure counts and not how it is embedded in the space. It is important to notice, however, that the behaviour of the combinatorial counterpart of these operators is found only by having a graph with uniform edge lengths. Indeed, we are studying spatial derivatives, therefore length is significant.
Even though our simulations are in agreement with theory in almost every respect, further studies are needed to understand why our numerical implementations show the 1.2 error factor. As it is shown in the previous Chapter, the first hypotheses we came up with are already proven wrong.
Now that we know the spectrum of these operators, we can definitely use it to study some differential equations coming from specific problems. In particular, it would be interesting to use our code to analyze some physical problems with an external three dimensional potential. Moreover, it should be possible to couple this type of problem with three-dimensional ones and study the whole model. Knowing the spectrum can be useful to implement preconditioners and to produce better discretizations and approximations for problems and solutions.
We are confident that our studies and our code can be easily extended to handle more complex problems, like parabolic ones, and boundary conditions different from Neumann-Kirchhoff or Dirichlet. Coding improvements and further implementation ideas are proposed in Section 7.10.

Chapter 7

C++ Implementation

7.1 Repository reference

This Chapter is devoted to presenting the main aspects regarding the C++ implementation involving some of the topics developed through this exposition. The latest version of the code can be found at the GitHub repository

`https://github.com/sabba11/PACS_FEMG_Abbate_Di_Primio`.

Configuration and compilation instructions can be found in `README.md`, thus they will not be reported hereafter. The core of the code is FEMG (Finite Element Methods on Graphs), a whole library devoted to solve elliptic problems on quantum graphs. The other parts of the code build the input data and compute some specific simulations. In the following the main principles behind the code will be explained, but this Chapter is not intended to be a detailed documentation of the code. In that regard, one can refer to the Doxygen routine included in the GitHub repository.

7.2 Dependencies

The library FEMG depends on several external libraries. They are listed here, with a brief explanation of their purpose in this work.

- GetFEM++ (ver. 5.3 or higher): finite element library. Main dependency of the code, exploited to assembly and solve differential problems on graph domains. It includes the numerical linear algebra library Gmm++.
- Linear Algebra PACKage (LAPACK): numerical linear algebra library. Used to exploit specific numerical methods to compute eigencouples of a generalized eigenvalue problem (QZ algorithm). Gmm++ provides a LAPACK-specific interface.
- Boost Graph Library (BGL, ver. 1.63 or higher): graph library. Used to handle folder management when exporting solutions.
- BGLgeom: old APSC project by I. Speranza and M. Tantardini to handle geometric and topological properties of graphs. Used to convert text files to graph data files (`.pts` files).

We have decided to employ these libraries because we needed both a way to deal with graph-like objects and both a way to implement finite element methods. We found out that using the format `.pts`, built by BGLgeom, was an easy way to produce input data for our problems. Indeed it was already used in some other works, as in the MANworks implementation made by Domenico Notaro and Stefano Brambilla, which deals with differential three-dimensional problems coupled with problems on graphs. This is very useful to know because we implemented our code so that an external three-dimensional potential could be imposed. For this reasons, those libraries were a good starting point to implement a generic code open to future improvement both in the definitions of more complex problems on more complex graphs, both to the implementation of coupled problems.
Furthermore, post-processing routines are mainly done in MATLAB, exploiting the GetFEM++-MATLAB interface.

7.3 Computational approach to differential problems

Let $\mathcal{P}$ denote a generic differential problem, where a set of unknowns $\mathcal{S}$ is to be found and a set of known coefficients $\mathcal{C}$ is given. The code operates according to logic shown in the flow chart of Figure 7.1.

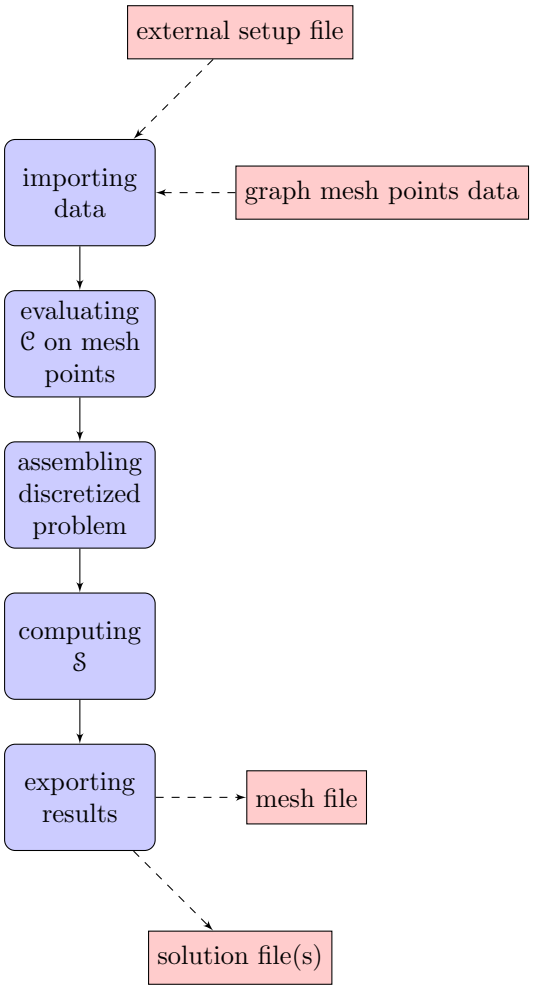


Figure 7.1: Flow chart illustrating the steps followed to solve a differential problem

The aim of the C++ work is to provide an interface to implement, in principle, a solver for any kind of differential problem on a graph. In particular, the library contains code to solve elliptic eigenvalue problems (with Neumann-Kirchhoff and homogeneous Dirichlet conditions) and elliptic differential problems (with Neumann-Kirchhoff, Dirichlet or mixed conditions).
The FEMG library is based on object-oriented programming. As all steps presented in Figure 7.1 are common to all kinds of problems, the computational counterpart of the core idea is to define an abstract differential problem (of general nature, not necessarily elliptic) on a quantum graph as an abstract class. In the code, this is represented by the header `include/quantum_graph_problem.hpp` (and the related source file in `src/` folder). Here is a brief excerpt (some detailed Doxygen comments have been deleted for readability).

```
class quantum_graph_problem {
  public:
    //! Virtual method to set up the problem and various parameters.
    virtual void init(int argc, char *argv[]) { //definition...
    }

    //! Method to evaluate known coefficients on the extended graph nodes.
    virtual void set_coefficients(const vector_function_type & f_vec, const vector_string_type &
    s_vec) = 0;

    //! Virtual method to assembly problem matrices and vectors.
    virtual void assembly(void) = 0;
```

```
32
    //! Solves the discrete problem.
34  virtual bool solve(void) = 0;

36  //! Exports the solution for external postprocessing.
    virtual void sol_export(void) = 0;
38
    //! Virtual destructor.
40  virtual ~quantum_graph_problem() {}

42  protected:
      //! Protected constructor.
44    quantum_graph_problem(void) :
        mimg(meshg), mf_Ug(meshg), mf_coeffg(meshg) {}
46
    // other protected members...
48 }
```

The structure of the public methods reflects the flow chart of Figure 7.1 and they are intended to be called in the presented order by the final executable. In particular

1. `init` takes terminal input and imports from a file all the necessary data and input parameters to set up the problem;
2. `set_coefficients` builds the appropriate data structures to handle given coefficients;
3. `assembly` builds the discrete problem matrices and vectors;
4. `solve` solves said problem imploying a suitable numerical algorithm;
5. `sol_export` exports solution data for further elaboration.

Several of the aforementioned methods are pure virtual as they have to be designed *ad hoc* with respect to the family of differential problems they should solve. We shall investigate their functionality in deeper detail in the following Sections. The class also possesses several protected data members to save quantum graph specifications which, again, are needed regardless the problem to solve, such as mesh, integration method, finite elements or data structures to save matrices and vectors of the discrete problem. Any particular differential problem is a derived class, child of `quantum_graph_problem` (via public inheritance). In this work, there are two main examples, the eigenvalue problem class `eigen_problem`, contained in the header `include/eigen_problem.hpp`, and the elliptic differential problem `elliptic_problem`, contained in the header `include/elliptic_problem.hpp`. They possess source files in the `src/` folder with the same name, but in the `.cpp` format. In the following, by "elliptic problem" we denote a problem of the kind represented by Equation (3.1), whereas by "eigenvalue problem" we denote a problem of the kind represented by Equation (3.5), up to different vertex condition choices.

7.4 Input data

The `init` method is virtual to allow customization, but it is already defined inside the `quantum_graph_problem` class. It executes a sequence of protected pure virtual methods, as shown below.

```
virtual void quantum_graph_problem::init(int argc, char *argv[]) {
2   //1. Read the .param filename from standard input
    INPUT.read_command_line(argc, argv);
4
    //2. Import data (algorithm specifications, boundary conditions,...)
6   import_data(); //pure virtual

8   //3. Build mesh for the graph
    build_mesh(); //pure virtual
10
    //4. Set finite elements and integration methods
12  set_im_and_fem(); //pure virtual

14  //5. Set default values for coefficients
    set_default_coefficients(); //pure virtual
16
    return;
```

```
}
```

The first action should be to read all the input data from a file, that has the format `.param` (but it is essentially a text file with a set format). This file is a sequence of lines of the type

```
TAG = VALUE
```

for which GetFEM++ provides methods to quickly import its contents to the C++ environment. To understand what must be inserted in such file, one should investigate how GetFEM++ handles specifications about, for instance, mesh convexes, finite elements or quadrature methods. GetFEM++ operates using strings called descriptors. Let us only provide a simple example, which can be taken as a basic paradigm for all similar cases. The integration method (which is the quadrature rule used to compute approximate integrals) is coded in the class `mesh_im` that has the constructor

```
getfem::mesh_mim(getfem::mesh mh)
```

which creates a structure linking the method to the mesh passed as argument. However, since there exists a large number of integration rules, it is necessary to specify which one is to be used. The class has the function member

```
void mesh_im::set_integration_method(getfem::pintegration_method ppi)
```

which sets the integration method represented from the descriptor `ppi` to the whole mesh (actually, it is also possible to select different quadrature rules on different convexes of the mesh using an overloaded version of the aforementioned method). Finally, the descriptor can be initialized as

```
getfem::pintegration_method ppi = getfem::int_method_descriptor("name of method")
```

where `"name of method"` is selected between a list of possible choices, including `IM_GAUSS(k)` (with k odd integer less than 99) or `IM_NC(n,k)` (with n and k integers), representing respectively Gauss and Newton-Cotes methods. This descriptor is imported through the `.param` file, and the `init` routine should set up the integration method. Analogous considerations hold for the choice of finite elements and for the nature of mesh convexes. The full lists of descriptors can be found inside GetFEM++ documentation. The collection of such descriptors (plus several custom ones) is stored in a struct called `descr_qg`, defined in the header `include/descr_qg.hpp`. The struct allocates space not only for GetFEM++-specific descriptors, but also for several custom descriptors, which turned out to be useful to let the user select different configurations (a complete list is in Appendix A). Following the same logic of `quantum_graph_problem`, `descr_qg` is an abstract struct, containing the descriptors common to all kinds of problems. Its children integrate the structure with the remaining, problem-specific, descriptors. Thus, for the `eigen_problem` and `elliptic_problem` cases, there are corresponding `eigen_descr_qg` and `elliptic_descr_qg` structs, children of `descr_qg` and defined in the headers with their same name, containing all the needed descriptors. Once all descriptors have been imported, the mesh is built reading a `.pts` file, which is essentially the list of all points of the discretized edges, with boundary condition data at the two extremes. Details on how to write this type of files are contained in the Doxygen documentation. Next, the GetFEM++ descriptors set integration and finite element methods, as anticipated before. Finally, known coefficients are set to a default value, in such a way that if the function `set_coefficients` is not called, a default problem is solved. For instance, the elliptic problem

$$-\operatorname{div}(p(x)\nabla u) + v(x)u(x) = f(x)$$

defaults to

$$-\Delta u = f(x)$$

if the coefficients $p(x)$ and $v(x)$ are not set. The source term f is instead mandatory.

7.5 Setting coefficients

The method `set_coefficients` should take care of known parameters of the differential problem having some kind of default value. The source term of an elliptic problem, instead, has the dedicated method `set_source` in the `elliptic_problem` class and it is the only mandatory coefficient to set. The function exploits the `<functional>` header to evaluate `std::functions` (whose declarations are directly in the `main` source file) on the mesh points. There are two methods to perform this action:

1. By direct evaluation, so that given a function g and a point $\mathbf{x}$, $g(\mathbf{x})$ is computed.

2. By what we have called "circular means", which are inspired by an application to cardiovascular systems. We think of each edge as an axis of a cylinder of a given radius (radius data must be read from file, see `IMPORT_RADIUS` and `RFILE` entries in Appendix A), and we evaluate the function at each point of the mesh by taking the arithmetic mean of a custom number of points equally distributed across the boundary of the (circular) cross section of the cylinder. To do so, an overloaded version of `set_coefficients` is implemented, taking as additional argument the number of points to take the mean on.

In both cases, a vector of functions and a vector of labels (strings) must be given as parameters. The labels determine the "role" of each coefficient and are problem-specific keywords, listed in Appendix A. The routines are similar across the two specific cases implemented in this work, however, since the labels are problem-specific, we preferred to keep this function pure virtual. The function `set_source` shares the same logic, except no labels are needed.

7.6 Assembly procedures

Assembly routines are called by the method `assembly`. As far as elliptic operators of the usual kind are concerned, recalling the theory developed in Chapter 3, stiffness and mass matrices (possibly weighted) are needed to build the discrete Hamiltonian operator. To achieve this, GetFEM++ provides a large amount of assembly routines. In particular, the documentation distinguishes between high-level and low-level procedures: the former introduces an *ad hoc* language (called weak form language) to implement very general weak formulations of boundary value problems, while the latter is designed mostly for linear problems. GetFEM++ documentation reports that, in general, the high-level assembly achieves better computational performance, but also that in the simple linear cases the low-level procedure may be faster. For this reason, and also to write clearer and simpler code, low-level assembly procedures have been employed. The paradigm of these methods is

```
template<typename MAT, typename VECT>
inline void getfem::asm_method (MAT &M, const mesh_im &mim, const mesh_fem &mf,
    const mesh_fem &mf_data, const VECT &A, const mesh_region &rg = mesh_region::all_convexes())
```

assembling data into the matrix type structure `M`, using the weights given in the vector type structure `A` and integrating on the mesh region specified in the optional argument `rg` (it defaults to the whole mesh). Mesh regions are a GetFEM++ tool which constitutes an easy way to keep track of privileged sets of convexes (or convex faces, which in the 1D case are points). When the mesh is imported, letting M be the number of branches in the original graph, $M+2$ or $M+3$ distinct mesh regions are created, labeled from 0 to $M+1$ or $M+2$. In particular:

- The mesh region labeled i, with $0 \leq i \leq M-1$, contains all convexes whose endpoints are both artificial discretization nodes lying on branch i.
- The mesh region labeled M contains all convexes whose endpoints are formed by one real point and one discretization node.
- The mesh region labeled $M+1$ contains all convex faces (i.e. nodes) of the original graph on which a Neumann-Kirchhoff boundary condition holds.
- The mesh region labeled $M+2$ contains all convex faces (i.e. nodes) of the original graph on which a Dirichlet boundary condition holds.

Mesh regions $M+1$ and $M+2$ are created only if the corresponding boundary condition is present. Methods used to build the vectors needed to setup the discrete problem have a paradigm almost identical to the one presented above, with the only difference that they assemble data into vectors instead of matrices. In particular:

- The known term in the discrete version of an elliptic problem (namely the right hand sides of Equations (3.3)) are assembled through a GetFEM++ routine called `getfem::asm_source_term`.
- This same method is used to assembly Neumann-Kirchhoff boundary conditions, simply by passing as arguments the given data and the corresponding mesh region.
- A slightly different approach is needed to assembly Dirichlet boundary conditions, although the method paradigm is again very similar. Indeed, Dirichlet conditions are coded directly modifying the final linear system to be solved, in such a way that the solution will satisfy them.

7.7 Solvers

Once the discrete problem is fully assembled, the method `solve` should compute its solution. In the implementation of this method in the derived classes, a suitable numerical method is chosen, according to the custom descriptor `COMP_METHOD` specified in the `.param` file. The GetFEM++ library relies on the numerical linear algebra library Gmm++, whose methods are used as solvers for the elliptic problem. The library Gmm++, however, do not contain all the solvers needed for the eigenvalue problem. In particular the QR algorithm is implemented, but it returns only eigenvalues and Schur vectors, not eigenvectors. To overcome this issue Gmm++ provides an interface to exploit some subroutines of LAPACK (and BLAS) library. Taking advantage of this device we implement the QR algorithm using:

```
gmm::geev_interface_right(dense_A, eigvals, eigvects);
```

However as was denoted in Section 3.2 differential problems always produce a generalized eigenvalue problem, having the mass matrix in the right hand size. For this reason an implementation of the QZ algorithm is advisable, but Gmm++ does not provide a version of it, neither the interface with any suitable LAPACK subroutine. Hence we implement the QZ algorithm using dggev LAPACK subroutine in the same way in which Gmm++ implement the interface with the dgeev routine. An external "C" linkage for the function is made

```
extern "C" {void dggev_( char* jobvl, char* jobvr, long unsigned int* na, double* a, long unsigned
     int* nb, double* b, long unsigned int* lda, double* wr, double* wi, double* wd, double* vl,
    long unsigned int* ldvl, double* vr, long unsigned int* ldvr, double* work, long int* lwork,
    long int* info );}
```

and afterwards all the needed conversions and initializations are made before calling it.
The particular numerical methods employed to solve elliptic and eigenvalue problems have already been roughly introduced in Chapter 3, thus they won't be covered here in much detail. Data about the computational performance are saved in a data structure of type `std::vector<std::pair<std::string, double>>`, called `log_data` (protected member of class `quantum_graph_problem`). Specifically, in the proposed derived classes, information about computational time, number of iterations, stopping criterion (namely if the algorithm reached the maximum number of iterations or achieved desired error accuracy), condition number of the matrix of the coefficients of the linear system (of course, the last three parameters are saved to `log_data` depending on the type of adopted solver - iterative or direct).

7.8 Exporting results

Finally, the method `sol_export` should export the obtained results for further elaboration. That includes all solution files and all mesh files, with information on how degrees of freedom are numbered, and how regions are composed. The collection of this (text) files is saved in a subdirectory of the folder located at the path `OUTPUT` given in the `.param` file. In the proposed derived classes, the final export path is built using some characteristics of the problem. Precisely, the final path will be `OUTPUT/export/N point-mesh/COMP_METHOD/` where `N` is the number of vertices in the extended graph. A check for existing duplicate folders is run when exporting data, so that conflicts are avoided.

7.9 Result postprocessing

The execution of the main executable ends with the exporting phase, as shown in the flow chart of Figure 7.1. In the `test_problem` folder (which is where the `main.cpp` source files are stored and the final executables generated) there is a collection of MATLAB routines to visualize and elaborate the exported results. The numerical experiments illustrated in Chapter 4 are analyzed using these procedures. The folder `test_problem/matlab_functions` contains auxiliary functions, while the main scripts are `FEMG_eigen_postprocessing.m` (contained in the `eigen/` subdirectory) and `FEMG_elliptic_postprocessing.m` (contained in the `elliptic/` subdirectory). All MATLAB routines rely on the GetFEM++-MATLAB inteface. These two routines share a similar logic: they are divided in smaller sections to be run one at a time to reduce the computational costs and let the user choose what type of analysis to conduct. Their aim is briefly presented hereafter.

`FEMG_elliptic_postprocessing.m`

0. Section 0 simply contains a directive to clear the workspace in such a way that a new set of solution files can be read. The user can select any number of variables from the current workspace and avoid their deletion (for example to compare multiple simulations).
1. Section 1 sets the paths to read solution data and to execute auxiliary functions (both custom and from the GetFEM++ interface).
2. Section 2 reads the solution files and stores the imported data in suitable structures.
3. Section 3 plots the computed solution over the graph.
4. Section 4 computes from the approximated solution the values of Dirichlet and Neumann-Kirchhoff boundary conditions. In particular, the latter conditions cannot be satisfied exactly by the numerical solution (however, they are asymptotically satisfied as the mesh is refined).
5. Section 5 is left for other customary postprocessing routines.

`FEMG_eigen_postprocessing.m`

0. Same as `FEMG_elliptic_postprocessing.m`.
1. Same as `FEMG_elliptic_postprocessing.m`.
2. Same as `FEMG_elliptic_postprocessing.m`.
3. Section 3 plots the computed spectrum sorted by ascending order (against their index).
4. Section 4 plots a number of eigenvectors over the graph. The user may choose which eigenvectors to plot.
5. Section 5 calls a function to count how many times an eigenvector changes sign across the graph.
6. Section 6 computes from the approximated solution the values of Dirichlet and Neumann-Kirchhoff boundary conditions. In particular, the latter conditions cannot be satisfied exactly by the numerical solution (however, they are asymptotically satisfied as the mesh is refined).
7. Section 7 compares the eigenvalue trend of the discretized Laplace operator with the one of the combinatorial Laplace matrix (Definition 1.5).
8. Section 8 is left for other customary postprocessing routines.

For further details one can check the comments in both files.

7.10 Future work

There are plenty of possible extensions to the work that has been done until now. The following are several ideas.

- Shift from the low-level assembly procedures to the high-level ones, yielding more generality and better computational performance for more complex problems (possibly nonlinear);
- Study parabolic problems (as child of `elliptic_problem` or as a separate child);
- Investigate other types of boundary conditions;
- Implement a parallel version of the code;
- Make the process of `.txt` graph data file creation more user-friendly;
- Modify the code in such a way that functions without a global analytical form (for example, defined piecewise on the edges of the graph) can be accepted as parameters of the differential problems;

7.11 Example of code usage

This Section aims to be a tutorial to build, solve and analyze a differential problem on a graph. To this end, its contents are organized in steps.

Step 0. Pre-compilation setup

It is assumed that the graph domain Γ and the strong form of the differential problem to be solved are known. Navigate to the `test_problem` folder, and select the subfolder corresponding to the type of problem to be solved. Therein, modify if necessary the `main.cpp` defining the source and potential terms after the `main` scope. They are functions of a `bgeot::base_node` object named `p`, whose coordinates can be accessed with `p[0]`, `p[1]` and `p[2]` (if existent). In the following steps, it is assumed that the code contained in the repository has been successfully compiled using `make all`. Further information on compilation instructions can be found in the `README.md` file contained in the main directory of the GitHub repository.

Step 1. Creating a text file

The first step is to build a `.txt` file containing vertex, edge and boundary condition data. For our test cases (i.e. the types of graph topology illustrated in Chapter 4), a collection of `.m` MATLAB routines is contained in the folder `data_builder/graph_maker`. Each of them is able to create (by simply running them) a text file containing a list of lines (one per edge) of the form

```
x1 y1 (z1) x2 y2 (z2) BC_TAG1 BC_VALUE1 BC_TAG2 BC_VALUE2
```

denoting coordinates and vertex conditions imposed at source (label 1) and target (label 2). Remark that the z-coordinates may or may not be present.
A set of premade `.txt` files is already available in the folder `data_builder/data/txt_files`.

Step 2. Creating a `.pts` file

The programs `pts_builder_h` and `pts_builder_N`, contained in `data_builder/graph_maked`, are able to read text files in the format specified in Step 1 and elaborate them to create a `.pts` file, which contains data about the discretized graph. Information about how these file are formatted are contained in the Doxygen documentation of the code. The two aforementioned programs are able to create h-type and N-type meshes respectively (check Section 4.4 for details) and can be called with the commands

```
$ ./pts_builder_N N path_to_input_txt path_to_output_pts
$ ./pts_builder_h h path_to_input_txt path_to_output_pts
```

to create the `.pts` file in the specified path. A set of premade `.pts` files is already available in the folder `data_builder/data/pts_files`.

Step 3. Setting up a `.param` file

Next, the descriptors have to be chosen. This is done through the `.param` file (which are basically renamed text files). The complete list of parameters is given in Appendix A. In particular, the following lines

```
OUTPUT = path_to_export_folder
MESH_FILEG = path_to_pts_file
```

denote where results are to be saved and where graph data is to be found respectively. Several `.param` files are already filled in with the necessary information and are located in the subfolders of `test_problem/eigen` and `test_problem/elliptic`.

Step 4. Executing the main routine.

The developed code is able to solve eigenvalue problems and standard elliptic boundary value problems, of the kind seen in the previous Chapters. To each kind different executables are devoted, contained in the subfolders of `test_problem/eigen` and `test_problem/elliptic`, respectively. The program can be run via the command

```
$ ./main path_to_param_file
```

Computed results will be found in the specified output directory.

Step 5. Elaborating the results.

Postprocessing of the results can be done through the `.m` routines contained in the subdirectories `test_probem/eigen` and `test_problem/elliptic`. Usage is briefly explained in Section 7.9, further details are contained in the comments of the MATLAB routines.

Appendices

Appendix A

A.1 Valid tags for `.param` files

Tags	Description	Valid values	Problem-specific?
`OUTPUT`	Sets output directory to export results.	A directory path.	No
`MESH_FILEG`	Sets file to read to import mesh data.	A path to an existing `.pts` file.	No
`MESH_TYPEG`	[GetFEM++] Sets convex type.	See GetFEM++ doc.	No
`FEM_TYPEG`	[GetFEM++] Sets FEM type.	See GetFEM++ doc.	No
`FEM_TYPEG_DATA`	[GetFEM++] Sets FEM type for known data.	See GetFEM++ doc.	No
`IM_TYPEG`	[GetFEM++] Sets quadrature rule.	See GetFEM++ doc.	No
`IMPORT_RADIUS`	Sets if radii data should be imported.	1 or 0.	No
`RFILE`	Sets file to read to import radii data.	Path to existing `.txt` file.	No
`COMP_METHOD`	Select computational method.	`QR`,`QZ` or `LU, CG, QMR, GMRES, BiCGSTAB`	Yes (eigen/elliptic)
`TOL`	Set tolerance for iterative solvers.	Positive real value.	Yes (eigen/elliptic)
`BY_ITERATION`	Sets if maximum number of iterations is given.	1 or 0.	Yes (eigen/elliptic)
`MAX_ITER`	Sets maximum number of iterations.	Positive integer value.	Yes (eigen/elliptic)
`RESTART`	Sets restarts for the GMRES method.	Positive integer value.	Yes (elliptic)

Table A.1: Valid tags and values for setup files.

A.2 Valid labels to set coefficients

A.2.1 `eigen_problem` case

The problem

$$-\operatorname{div}(a(x)\nabla u) + v(x)u(x) = \lambda p(x)u(x)$$

has the coefficients:

1. $a(x)$ with label `left_weights`;
2. $v(x)$ with label `potential`;
3. $p(x)$ with label `right_weights`.

A.2.2 `elliptic_problem` case

The problem

$$-\operatorname{div}(a(x)\nabla u) + v(x)u(x) = f(x)$$

has the coefficients:

1. $a(x)$ with label `left_weights`;
2. $v(x)$ with label `potential`;

Appendix B

B.1 On the Laplace matrix spectral comparison

The following Tables report the numerical values of the eigenvalues computed to draw the comparison between the extended and the combinatorial Laplace matrix in Subsection 4.2.4.

Star-shaped graph	
Combinatorial Laplace matrix	**Extended Laplace matrix**
0	0
1	2.4674
1	2.4674
1	2.4674
5	9.8704

Graphene-like graph	
Combinatorial Laplace matrix	**Extended Laplace matrix**
0	0
0.1578	0.1571
1	0.9257
1	1.0966
1	1.0966
1.4931	1.5608
3	3.5808
3	4.3867
3	4.3867
3.5069	4.7504
4	7.5370
4.8422	9.8704

Tree graph			
Comb. Laplace matrix	**Fixed edge length, decaying angle**	**Fixed angle ($\pi/4$), decaying edge length**	**Fixed angle ($\pi/4$), decaying edge length (*)**
0	0	0	0
0.0968	0.1155	0.3303	0.7426
0.2679	0.3788	1.1187	1.5486
0.2679	0.3788	2.2207	5.4148
0.4965	0.7074	2.2207	10.2861
1	2.4676	4.2579	12.4172
1	2.4676	6.7028	12.4172
1	2.4676	11.1139	16.3612
1	2.4676	16.5891	26.6866
1.7356	2.4676	19.2949	33.2843
2.1939	2.4676	22.2269	45.3512
3.5767	5.2936	22.2269	61.8818
3.7321	6.3829	22.2270	61.9300
3.7321	6.3829	22.2270	81.1793
4.7093	7.8524	29.3216	99.9868
5.1912	9.8736	34.6140	112.1890
Fixed angle ($\pi/6$), decaying edge length	**Fixed angle ($\pi/6$), decaying edge length (*)**	**Decaying angle, decaying edge length**	**Decaying angle, decaying edge length (*)**
0	0	0	0
0.3303	0.7426	0.3303	0.7426
1.1187	1.5486	1.1187	1.5486
2.2207	5.4148	2.2207	5.4148
2.2207	10.2861	2.2207	10.2861
4.2579	12.4171	4.2579	12.4172
6.7028	12.4171	6.7028	12.4172
11.1139	16.3613	11.1139	16.3612
16.5891	26.6866	16.5891	26.6866
19.2949	33.2843	19.2948	33.2843
22.2269	45.3512	22.2269	45.3512
22.2269	61.8818	22.2269	61.8818
22.2269	61.9299	22.2269	61.9300
22.2269	81.1793	22.2269	81.1793
29.3216	99.9868	29.3216	99.9867
34.6139	112.1890	34.6139	112.1890

Table B.1: Computed eigenvalues for various graph topologies.

Bibliography

[1] Albert-László Barabási, Reká Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[2] Alfio Quarteroni, Riccardo Sacco, Fausto Saleri. *Numerical Mathematics*. Texts in Applied Mathematics. Springer, 2nd edition, 2006.

[3] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 3rd edition, 2006.

[4] Gene H. Golub, Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 4th edition, 2013.

[5] Gregory Bertolaiko, Peter Kuchment. *Introduction to Quantum Graphs*. Mathematical Surveys and Monographs. American Mathematical Society, 2013.

[6] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[7] Mario Arioli, Michele Benzi. A finite element method for quantum graphs. *IMA Journal of Numerical Analysis*, 38:1119–1163, June 2017.

[8] Sergiu Moroianu. Weyl laws on open manifolds. *Mathematische Annalen*, 340, 11 2003.

[9] K. Naimark and M. Solomyak. Eigenvalue estimates for the weighted laplacian on metric trees. *Proc. London Math. Soc. (3)*, 80:690–724, 05 2000.

[10] K. Naimark and M. Solomyak. Geometry of sobolev spaces on regular trees and the hardy inequalities. *Russian Journal of Mathematical Physics*, 8, 07 2001.

[11] Michael Solomyak. On eigenvalue estimates for the weighted laplacian on metric graphs. *Nonlinear Problems in Mathematical Physics and Related Topics I*, 02 2002.

[12] Michael Solomyak. On the spectrum of laplacian on regular metric trees. *Waves in Random Media - WAVE RANDOM MEDIA*, 14, 01 2004.

[13] Matt Stevenson. Weyl's law. *Notes for a talk held at University of Michigan*.