Documentation for graphcurvature.py

David Cushing

George Stagg

April 15, 2016

Abstract

This documentation gives an overview of the file graphcurvature.py.

1 Introduction

Let $G = (V, E)$ be a finite simple graph. For any vector (function) $f : V \to \mathbb{R}$ and any vertex $x \in V$, the Laplacian Δ is defined via

$$\Delta f(x) := \frac{1}{\mu(x)} \sum_{y, y \sim x} (f(y) - f(x)), \tag{1.1}$$

where $\mu : V \to \mathbb{R}$ is a positive measure on V. When $\mu(x) = 1$, for any $x \in V$, we call Δ the non-normalized Laplacian. When $\mu(x) = d_x := \sum_{y, y \sim x} 1$, for any $x \in V$, we call Δ the normalized Laplacian.

For any two functions $f, g : V \to \mathbb{R}$, we define two operators Γ and Γ_2 as follows:

$$2\Gamma(f, g) := \Delta(fg) - f\Delta g - (\Delta f)g, \tag{1.2}$$

$$2\Gamma_2(f, g) := \Delta(\Gamma(f, g)) - \Gamma(f, \Delta g) - \Gamma(\Delta f, g). \tag{1.3}$$

Note that $\Gamma(f, f)(x)$ and $\Gamma_2(f, f)(x)$ can be represented as matrices defined on the two-ball of x and acting on f restricted to the two-ball. We denote these matrices as $\Gamma(x)$ and $\Gamma_2(x)$.

Definition 1.1. *Let $\mathcal{K} \in \mathbb{R}$ and $\mathcal{N} \in \mathbb{R}_+$. We say that the graph $G = (V, E)$ satisfies the curvature-dimesion inequality (CD inequality) $CD(\mathcal{K}, \mathcal{N})$, if for any $f : V \to \mathbb{R}$ and any $x \in V$, we have*

$$\Gamma_2(f)(x) \geq \frac{1}{\mathcal{N}}(\Delta f(x))^2 + \mathcal{K}\Gamma(f)(x). \tag{1.4}$$

Here, $\mathcal{K}$ is called a lower Ricci curvature bound of $G = (V, E)$, and $\mathcal{N}$ a dimension parameter. At a vertex $x \in V$, the precise $\mathcal{N}$-dimensional Ricci curvature lower bound $\mathcal{K}_{\mathcal{N}}(G, x)$ *is defined as the largest $\mathcal{K}$ such that (1.4) holds for a given $\mathcal{N}$.*

2 The Programme

2.1 Calculating the curvature

We enter graphs into Python via their adjacency matrix. For example, a triangle, i.e. the complete graph on 3 vertices, is entered as follows:

```
>>> T = [[0,1,1],[1,0,1],[1,1,0]]
```

Similarly the Petersen graph would be entered as:

```
>>> P=[[0,1,0,0,1,1,0,0,0,0],[1,0,1,0,0,0,1,0,0,0],[0,1,0,1,0,0,0,1,0,0],
[0,0,1,0,1,0,0,0,1,0],[1,0,0,1,0,0,0,0,0,1],[1,0,0,0,0,0,0,1,1,0],
[0,1,0,0,0,0,0,0,1,1],[0,0,1,0,0,1,0,0,0,1],[0,0,0,1,0,1,1,0,0,0],
[0,0,0,0,1,0,1,1,0,0]]
```

The functions `curv_calc` and `curv_calc_norm` calculate the curvature at a specified vertex with respect to the non-normalised and normalised Laplacian, respectively. Note that the vertices are specified via their vertex numbers and that the enumeration starts from 0.

```
>>> curv_calc(T, 0)
2.5
>>> curv_calc_norm(T, 0)
1.25
>>> curv_calc(T, 0)
-1.0
>>> curv_calc_norm(T, 0)
-0.33
```

2.2 The matrices Γ and Γ_2

In this section we discuss the commands generating the matrices Γ and Γ_2. The matrices are viewed conveniently via the numpy command `array`.

Generally 4Γ and $4\Gamma_2$ instead of Γ and Γ_2 ensures that all the terms in the matrices with respect to the non-normalised laplacian are integer valued. Again, the vertex in these commands is specified by its vertex number. The commands are `fourGamma`, `fourGamma2`, `fourGammaNorm` and `fourGamma2Norm`.

```
>>> np.array(fourGamma(T, 0))
[[ 4 -2 -2]
 [-2  2  0]
 [-2  0  2]]
>>> np.array(fourGamma2(T, 0))
[[10 -5 -5]
 [-5  7 -2]
 [-5 -2  7]]
>>> np.array(fourGammaNorm(T, 0))
[[ 2. -1. -1.]
 [-1.  1.  0.]
 [-1.  0.  1.]]
>>> np.array(fourGamma2Norm(T, 0))
[[ 2.5  -1.25 -1.25]
 [-1.25  1.75 -0.5 ]
 [-1.25 -0.5   1.75]]
```

Note that the Γ- and Γ_2-matrices have a different vertex ordering than the original adjacency matrix. It is convenient to build up these matrices by rearranging the vertices in the order "centre, one-sphere, two-sphere". The command `mat_order` reveals this order for a specified vertex in form of the original vertex numbers.

```
>>> np.array(fourGamma(P, 0))
[[ 6 -2 -2 -2  0  0  0  0  0  0]
 [-2  2  0  0  0  0  0  0  0  0]
 [-2  0  2  0  0  0  0  0  0  0]
 [-2  0  0  2  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
```

```
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]]
>>> np.array(fourGamma2(P, 0))
[[18 -8 -8 -8  1  1  1  1  1  1]
 [-8  8  2  2 -2  0 -2  0  0  0]
 [-8  2  8  2  0 -2  0  0  0 -2]
 [-8  2  2  8  0  0  0 -2 -2  0]
 [ 1 -2  0  0  1  0  0  0  0  0]
 [ 1  0 -2  0  0  1  0  0  0  0]
 [ 1 -2  0  0  0  0  1  0  0  0]
 [ 1  0  0 -2  0  0  0  1  0  0]
 [ 1  0  0 -2  0  0  0  0  1  0]
 [ 1  0 -2  0  0  0  0  0  0  1]]
>>> mat_order(P, 0)
[0, [1, 4, 5], [2, 3, 6, 7, 8, 9]]
```

Note that you can also use this function to calculate the one-sphere and two-sphere of a given vertex.

Finally, the command `evs` calculates the eigenvalues of a given symmetric matrix. For example

```
>>> evs(fourGamma(T,0))
[0.  2.  6.]
```

3 Edge and Vertex weights

The functions `fourGammaFULL(W, MU, i)` and `fourGamma2FULL(W, MU, i)` calculate the matrices, defined in (1.2) and (1.3), with edge weight matrix W and a vector containing the vertex weights MU.

In this case the Laplacian assumes the form

$$\Delta f(x) := \frac{1}{\mu(x)} \sum_{y, y\sim x} w_{xy}(f(y) - f(x)), \tag{3.1}$$

with $w_{xy} = W(x,y)$ and $\mu(x) = MU(x)$.

For example, suppose we want to enter a triangle with edges weights 1, 2 and 3 and calculate its $4\Gamma_2$ matrices about each vertex. We would do this as follows:

```
>>> T = [[0,1,2],[1,0,3],[2,3,0]]
>>> MU = [1,1,1]
>>> np.array(fourGammaFULL(T, MU, 0))
[[ 6. -2. -4.]
 [-2.  2.  0.]
 [-4.  0.  4.]]
>>> np.array(fourGammaFULL(T, MU, 1))
[[ 8. -2. -6.]
 [-2.  2.  0.]
 [-6.  0.  6.]]
>>> np.array(fourGammaFULL(T, MU, 2))
[[ 10.  -4.  -6.]
 [ -4.   4.   0.]
 [ -6.   0.   6.]]
```

Note that if you choose the adjacency matrix for W then you obtain the Laplacian defined in equation (1.1).

4 Interactive website

In addition to the python programme there is an interactive website which allows you to draw a graph and to view the curvature of the graph at each vertex with respect to both the normalised and non-normalised Laplacian. Online instructions can be found on the webpage itself.

To play with it go to `http://teggers.eu/graph/`.