

Project: Heat kernel construction on cortical surface using geodesic distance

STAT 992: Statistical Methods in Signal and Image Analysis

Instructor: Moo K. Chung

Tulaya Limpiti

May 15, 2004

1 Introduction

In medical imaging, measured data always suffer from noise. One way to effectively increase the signal-to-noise ratio is to perform smoothing on the data, where data points are weighted-averaged with their neighbors. Smoothing can be viewed as low-pass filtering, since the high-frequency components are removed from the smoothed data. In brain imaging, the data points lies on the cortical surface, and the underlying functional data are naturally smooth, for example, cortical thickness, blood oxygenation level dependent, or neural activity patterns [7,1,2]. To estimate signal component from noisy observations, kernel-based smoothing is the most popular non-parametric estimator. Different kernels have been used in the smoothing process, such as uniform (square) kernel, Gaussian kernel, Epanechnikov kernel, and triweight kernel [16]. The most widely used kernel is Gaussian kernel, which is originally defined on an n-dimensional Euclidean space.

Nevertheless, in brain imaging our data of interest lies instead on a cortical surface, which is assumed to be a smooth two-dimensional Riemannian manifold [8]. Because of the convoluted nature of the cortical surface, two points on the surface that appear close together in a Euclidean sense can actually be very far apart, for example, when each of the two points lies on an opposite bank of a sulcus (see figure 1). Therefore, if we were to apply Gaussian kernel smoothing on this manifold using a kernel that is isotropic in Euclidean space, we may give large weights to data that are far away, and small weights to data that are closer. In other words, we would mistakenly put higher emphasis on the less correlated data in the resulting weighted average, which would be inappropriate. An alternative smoothing method is diffusion smoothing using diffusion (heat) kernel. It has been shown [6,8,11] that there is a tight relationship between heat kernel and Gaussian kernel. On geometric manifolds, heat kernel is a function of geodesic distance—shortest distance along the surface—instead of Euclidean distance. Geodesic distance provides a proper measure of correlations between data points.

In this report, we construct heat kernels based on geodesic distances of data points on the cortical surface and perform kernel smoothing using our kernels on cortical thickness measurement data. We note that our heat kernel is locally equivalent to an isotropic Gaussian kernel, and hence the amount of smoothness can be controlled by adjusting the bandwidth matrix of the kernel. The geodesic distances calculation in our kernel construction is carried out via dynamic programming. The generalization of the result to anisotropic kernels smoothing is possible, although it is beyond the scope of this project.

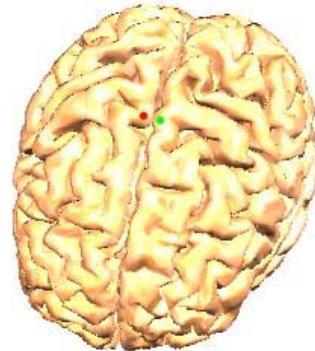


Figure 1: Cortical surface. The red and green dots are close in the Euclidean sense. But to get from one dot to the other requires a long traveling distance along the curve (down the sulcus and back up).

To represent the cortical surface, we have adopted the polygonal representation, commonly used in the computer vision community. The problem of finding minimal distances between arbitrary points on polyhedral surface has attracted interests of many researchers in the field of graph theory, robotics motion planning, computer-aided neuroanatomy, to name a few. Several algorithms have been developed. The first algorithm is that of Dijkstra [10] where he solved the problem on graphs in 1959. Sharir and Schorr [15] proposed the algorithm that finds the shortest path in $O(n^3 \log n)$, where n is the total number of nodes on the surface. However, they constrained themselves to consider the problem only on convex polyhedrons. Mitchell, Mount, and Papadimitriou came up with an improved algorithm with continuous Dijkstra structure and the computational complexity of $O(n^2 \log n)$ [12]. Other algorithms include those contributed by Chen and Han [5], and Wolfson and Schwartz [17]. Among these algorithms, dynamic programming (DP) finds itself become popular, due to its flexibility and its ease of implementation for discrete constraint sets. The complexity of DP is $O(n^2)$.

This report is organized as followed. In the next section we review some relevant backgrounds necessary to understand our proposed method. We start with the definition of Gaussian kernel and heat kernel and their uses in data smoothing and estimation in section 2.1. Riemannian manifold and its polygonal representation are explained in section 2.2, where as in section 2.3 we give a mathematical description of geodesic distance. Section 2.4 introduces the concepts of dynamic programming. In section 3 we give details of our proposed construction of isotropic heat kernel on the cortical surface using geodesic distance. Experimental results using cortical thickness measurement data are presented in section 4. Finally, we conclude with some discussions on our results in the last section.

Throughout the report, matrix and vector quantities are written in bolded-face uppercase and lowercase letters, respectively. Superscript $\{\cdot\}^T$ denotes matrix transpose, and superscript $\{\cdot\}^{-1}$ denotes matrix inverse.

2 Backgrounds

2.1 Gaussian kernel and Heat kernel

Generally, a *Gaussian kernel* is defined in an n -dimensional Euclidean space as

$$K_H(x) = (2\pi)^{-\frac{n}{2}} (\det HH^T)^{1/2} \exp\left(-\frac{x^T (HH^T)^{-1} x}{2}\right) \quad (1)$$

where $\mathbf{x} = [x_1, x_2, \dots, x_n]^T \in \Re^n$. The n -by- n matrix \mathbf{H} is called the *bandwidth matrix* of the kernel [6]. For isotropic kernel, $\mathbf{H} = \sigma^2 \mathbf{I}_n$, where \mathbf{I}_n is an n -by- n identity matrix and σ^2 is a user-defined positive number. With this bandwidth matrix, the isotropic kernel has the form:

$$K_\sigma(x) = \left(2\pi\sigma^2\right)^{-\frac{n}{2}} \exp\left(-\frac{\sum_{i=1}^n x_i^2}{2\sigma^2}\right) \quad (2)$$

The bandwidth matrix governs the amount of smoothing. There is a bias-variance tradeoff between small and large bandwidth. Small bandwidth corresponds to undersmoothing, which has small bias and large variance. On the other hand, oversmoothing, with large bias and small variance, results from large bandwidth selection [13].

In image analysis, we treat observations as a set of samples drawn from a continuous function, corrupted by noise. To separate the signal component from the noise component using isotropic Gaussian kernel smoothing, the kernel is convolved with the signal,

$$\mu(x) = \int_{R^n} K_H(x-y)f(y)dy. \quad (3)$$

Theoretically, Gaussian kernel is desirable because it is continuously differentiable, separable, and symmetric. In practice, care has to be taken in implementing a Gaussian kernel because of its infinite support. We use the discrete, truncated version of the kernel, as given in [6]:

$$\tilde{K}_H(x_i) = \frac{K_H(x_i)l_\Omega(x_i)}{\sum_{\forall x_i \in \Omega} K_H(x_i)} \quad (4)$$

where Ω is the constraint region. In nonparametric regression setting, this is called the Nadaraya-Watson estimator [7].

Given observations $Y(\mathbf{x})$, $\mathbf{x} \in M$, *heat kernel* $K_t(\mathbf{x}, \mathbf{y})$ on geometric manifolds is governed by the second-order differential equation $\frac{\partial f}{\partial t} = \Delta f$, with the initial condition $f(\mathbf{x}, 0) = Y(\mathbf{x})$. Δ is the Laplace-Beltrami operator. The solution of this PDE is

$$f(x, t) = \int_M K_t(x, y)f(y)dy \quad (5)$$

For the simple manifold, $M = \Re$, $K_t(\mathbf{x}, \mathbf{y})$ is equivalent to an isotropic Gaussian kernel with bandwidth $\sigma^2 = 2t$ [11]. In other words, the result of the diffusion of initial data Y for the time duration $t = \sigma^2/2$ is similar to the result of the convolution of an isotropic Gaussian kernel ($\sigma^2 = 2t$) with the data Y .

The parametric expansion of the heat kernel has the form [11]:

$$P_t^{(m)}(x, y) = (4\pi t)^{-1/2} \exp\left(-\frac{d^2(x, y)}{4t}\right) \cdot [\psi_0(x, y) + \psi_1(x, y)t + \dots + \psi_m(x, y)t^m] \quad (6)$$

Assuming t is sufficiently small and the point \mathbf{x}, \mathbf{y} are close together, the heat kernel can be approximated locally as [6]:

$$K_t(x, y) \approx (4\pi t)^{-1/2} \exp\left(-\frac{d^2(x, y)}{4t}\right) \quad (7)$$

It is obvious that the approximation resembles the form of isotropic Gaussian kernel with bandwidth $2t$. The only difference is that the Euclidean distance term in the exponent is replaced by geodesic distance. Therefore, geodesic distance turns the Riemannian manifold into a metric space, satisfying the usual properties, e.g., positivity, symmetry, and triangle inequality. Analytically, the heat kernel in (7) is the kernel we attempt to construct, of which the details are given in section 3.

One advantage of the diffusion smoothing is that it solves the problem usually encountered with Gaussian kernel smoothing at the boundary. Fortunately, the cortical surface is a compact manifold, and so the problem at boundaries is not applicable in our specific case.

Another drawback of kernel smoothing that has been pointed out in [16] is the difficulty in finding the appropriate value of the bandwidth to be used. One solution is to plot the integrated squared error (ISE) between the smoothed data and the original signal. The optimal bandwidth is identified as the global minima of the ISE plot. However, the original, noise-free signal is rarely known in advance, which makes the ISE idea only suitable for algorithm validations. Simonoff also suggests several other bandwidth selection methods. One interesting method that exploits the geometry of the manifold is called *local-varying bandwidth*, where distance from point x to its k^{th} nearest neighbor (for some fixed k) is used as σ^2 , results in more smoothing in the regions of low density. In this region, distance from x to the k^{th} neighbor would be large, and so as the bandwidth. Unfortunately this local-varying bandwidth does not give satisfactorily results, and some subtle features of the data are missed.

In practice, an alternative solution is called *iterated kernel smoothing* [6,7]. Argued by induction, an iterated kernel smoothing formula is

$$K_\sigma^{(m)} * f = \underbrace{K_\sigma * K_\sigma * \dots * f}_m = K_{\sqrt{m}\sigma} * f \quad (8)$$

This formula says that smoothing with large bandwidth to be achieved by iteratively apply smoothing using smaller bandwidth. We make use of this interesting property in our experiments in section 4.

2.2 Riemannian manifold and its polygonal representation

A *Riemannian manifold* (M, g) is a differentiable manifold with a family of smoothly varying, positive definite inner products $g = g_p$ defined on a tangent space $T_p M$ for each $p \in M$ [11]. The *Riemannian metric tensors* $\mathbf{G} = \{g\}_{ij}$ is given as

$$G_p(v, w) = \sum_{i,j} g_{ij}(p)v_i w_j \quad , \text{ where } \mathbf{v} \text{ and } \mathbf{w} \text{ belong to } T_p M.$$

Every manifold has its associated Riemannian metric, which enable us to define the length of vectors and curves on the manifold. That, in turn, allows the computation of the geodesic distance, as we cover in more details in the next subsection.

The 2-D Riemannian manifold of interest in our project is a two-dimensional cortical surface. The *polygonal representation* represents the surface by a set of connected planar polygons. The surface S is completely described by a set of F faces and V vertices, $S = \{S_F, S_V\}$. Each row of the V -by-3 vertex matrix S_V is the vector containing rectangular coordinates of a specific vertex $\{v_i\}_{i=1,\dots,V}$, indexed by the row number. For example, the coordinate of the 5th vertex (v_5) is in the 5th row of S_V . Each of the rows of the F -by-3 face matrix S_F contains a vector of three non-repeated indices of vertex that comprises the triangular face $\{f_i\}_{i=1,\dots,F}$. Figure 2 shows an example of the polygonal representation of a simple manifold.

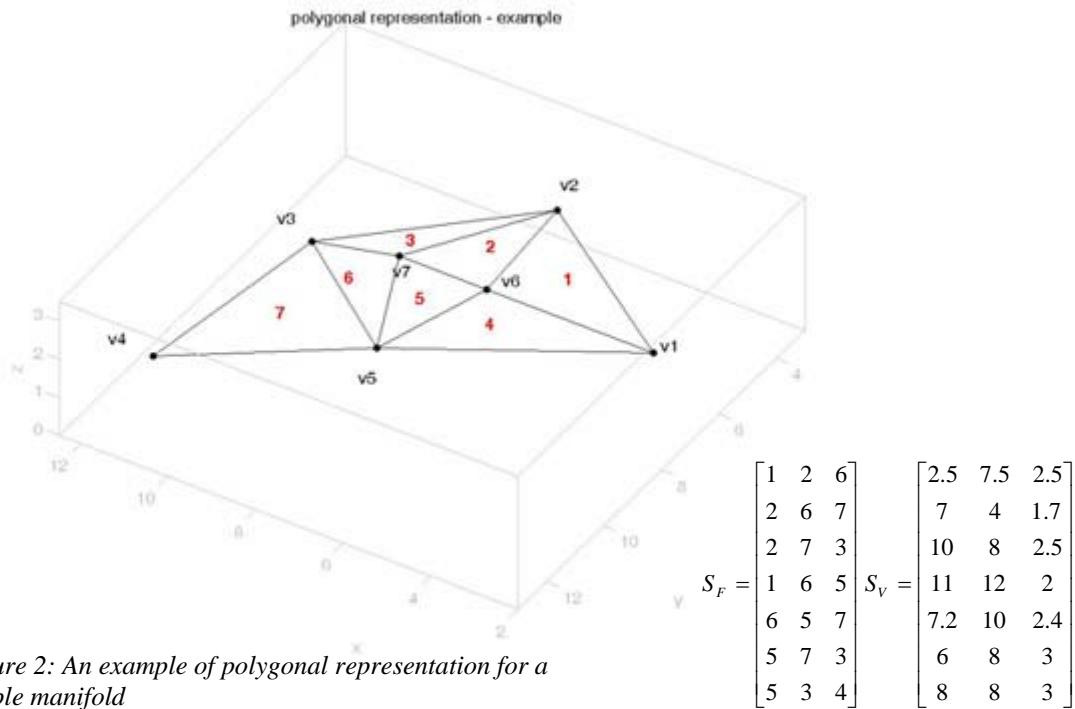


Figure 2: An example of polygonal representation for a simple manifold

It is known that to accurately represent a smooth curve of the cortical surface, an order of 10^4 vertices are sufficient. Since the cortical surface has a topology of a sphere, we have that $2V - F = 4$ in order for the surface to be topologically correct [6].

To acquire the face matrix and vertex matrix, a high-resolution MRI of the subject's brain is input into an automated software FreeSurfer [9], which segments the structural MRI data and reconstructs the 3-D topologically corrected triangular tessellation of the cortical surface. Figure 3 illustrates the left hemisphere of the cortical surface obtained from FreeSurfer.

3D cortical surface reconstruction from FreeSurfer- Left hemisphere

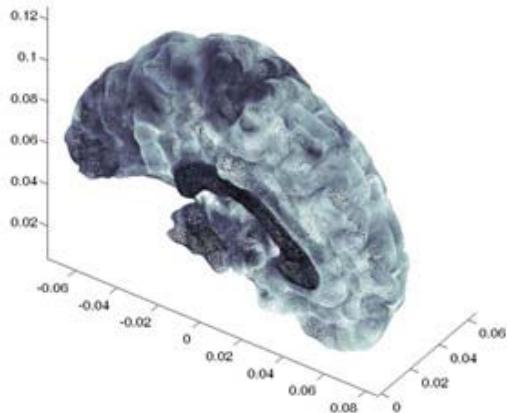


Figure 3: 3-D reconstruction of the cortical surface obtained from FreeSurfer

2.3 Geodesic distance

Let the curve $C \subset M$ connecting two points \mathbf{x} and \mathbf{y} be parameterized by $\gamma: [0,1] \rightarrow C$, With $\gamma(0) = \mathbf{x}$ and $\gamma(1) = \mathbf{y}$. Then, the length of the curve is

$$l(\gamma) = \int_0^1 g_p(\gamma'(t), \gamma'(t))^{1/2} dt \quad (9)$$

The *geodesic distance* $d(\mathbf{x}, \mathbf{y})$ is the shortest piecewise differentiable curve connecting \mathbf{x} and \mathbf{y} [11]. Analytically, $d(\mathbf{x}, \mathbf{y})$ can be obtained by simply minimizing the function $l(\gamma)$ for $\forall \gamma$. This is a calculus of variations problem, which at times can be cumbersome. The compactness of the manifold guarantees the solution of such problem exists [14]. In our specific case, however, by representing the cortical surface with its polygonal representation, we discretize the cortical surface and the corresponding functional data on the manifold, and hence put ourselves in the realm of optimization problems over finite state space, where vast body of literature exists. The next subsection gives a brief introduction to dynamic programming, one of the well-known tools in optimization we utilize in our kernel construction.

2.4 Dynamic programming

The dynamic programming technique is dated back to 1960s when Bellman pioneered the idea [3]. The key concept of dynamic programming rests on the *principle of optimality*, which states the following intuitive fact [4]:

Given an objective function

$$J = \sum_{k=0}^{N-1} J_k(x(k), u(k)) + \Psi(x(N))$$

where $x(k)$ is the state of the system and $u(k)$ is the decision to be optimized.

J_k is the cost at particular stage k of the optimization process, and Ψ is the terminal cost at the last stage.

Suppose $u^*(0), u^*(1), \dots, u^*(N-1)$ minimizes the cost function J , and let $x^*(k)$ be the *trajectory mate* of $u^*(k)$; i.e.,

$$x^*(k+1) = f(x^*(k), u^*(k), k)$$

Define a subproblem beginning at $x^*(k)$; $k \geq 0$, with the objective

$$J_s = \sum_{i=k}^{N-1} J_i(x(i), u(i)) + \Psi(x(N))$$

Then, $u^*(k), u^*(k+1), \dots, u^*(N-1)$ also minimizes J_s .

The idea of this principle is very simple. To exemplify, let's say we would like to find P_{AC}^* , a shortest path to travel from the origin city A to destination city C. Suppose that city B lies between A and C, and the shortest path from A to C passes through B. Then, it is obvious that The optimal path from B to C, denoted by P_{BC}^* , has to be a sub-path of P_{AC}^* (see figure 4). The principle of optimality can be proved by contradiction.

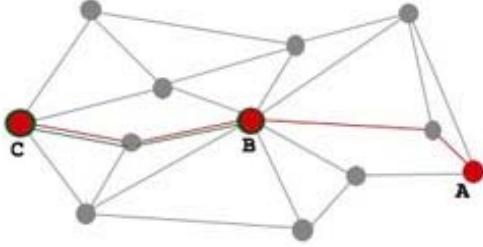


Figure 4: city travel example. P_{AC}^* is displayed in red, and P_{BC}^* is in green. It is clear that $P_{BC}^* \subset P_{AC}^*$.

In the past years, DP has found success in wide range of applications, both in deterministic and stochastic systems, for example, in inventory control, critical path analysis, scheduling, or even in gambling strategies [4]. The example in the previous paragraph is a simple version of the so-called *shortest path problem*, which use the total traveling distances between cities as its cost to be optimized. It is important to note that DP can be numerically solved in polynomial time, and is guaranteed to converge to a globally optimal solution. These desirable properties make dynamic programming a good candidate for our geodesic distance calculation. In section 3 we explain in details how geodesic distance computation of points on the cortical surface can be formulated within the dynamic programming framework.

3 Proposed kernel constructions

Recall that the isotropic heat kernel on the cortical surface is approximated as

$$\tilde{K}_\sigma(x_i) = \frac{K_\sigma(x_i)l_\Omega(x_i)}{\sum_{\forall x_i \in \Omega} K_\sigma(x_i)}; \quad K_\sigma(x, y) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{d^2(x, y)}{2\sigma^2}\right). \quad (10)$$

Note that although we are talking about heat kernel, t and σ can be used interchangeably.

In order to construct the truncated kernel, there are three parameters in equation (10) to be specified, namely σ , Ω , and $d(\mathbf{x}, \mathbf{y})$. Appropriate selections of these parameters are addressed in the construction steps below. Our heat kernel construction procedure can be divided into three steps:

Step 1: Constraint regions identification – selecting Ω .

Step 2: Geodesic distance calculation – finding $d(\mathbf{x}, \mathbf{y})$.

Step 3: Kernel weights computation – choosing σ .

Next, we explain each step in details.

3.1 Constraint regions identification

Let $N_m(i)$ denote a set of m^{th} -order neighbors of vertex v_i .

Definition 1: Vertex v_j is a *first-order neighbor* of vertex v_i if v_i and v_j share an edge.

Definition 2: Vertex v_j is a *second-order neighbor* of vertex v_i if vertex v_k is a first-order neighbor of both v_i and v_j ; and v_j is not already a first-order neighbor of v_i , for $\exists v_k \in N_1(i)$.

Definition 3: Vertex v_j is an m^{th} -*order neighbor* of vertex v_i if vertex v_k is a first-order neighbor of v_j and an $(m-1)^{\text{th}}$ -order neighbor of v_i ; and v_j is not already any lower-order neighbors of v_i , for $\exists v_k \in N_{m-1}(i)$.

In order to preserve the shape of the cortical surface, the vertices of the triangular tessellation given by FreeSurfer are non-uniformly spaced. The constraint region Ω_i of vertex v_i is defined to include up to 3^{rd} -order neighbors, as illustrated in figure 5. Mathematically, the constraint region is defined as the union of all triangular faces that its corresponding vertices are either v_i or its 1^{st} -, 2^{nd} -, and 3^{rd} -order neighbors, i.e.;

Let each face be represented as $f_k = [v_{k,1}, v_{k,2}, v_{k,3}]$. Then,

$$\Omega_i = \left\{ \bigcup_k f_k : v_{k,j} \in \{v_i \cup N_1(i) \cup N_2(i) \cup N_3(i)\}, \forall j = 1, 2, 3 \right\}, i = 1, \dots, V$$

The choice of the number of neighboring vertices to be included in Ω_i is empirical. If Ω_i is too large, we would oversmooth the data. On the other hand, if Ω_i is too small we would undersmooth. The extent of the constraint region depends largely on specific applications. Growing larger constraint region is achieved by adding more vertices from higher-order neighboring set.

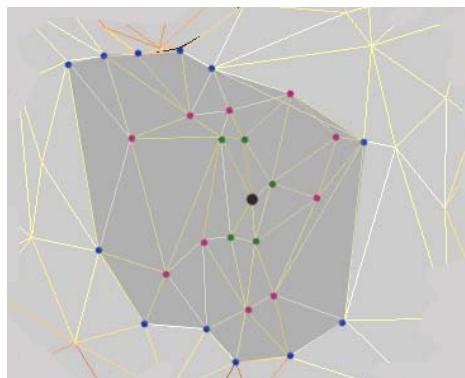


Figure 5: example of region B on the cortical surface (zoomed-in) for the truncated kernel. Green, red, and blue dots denote first, second, and third neighbors of the data point (black), respectively

If Ω_i is such that only the first-order neighbors are considered, we arrive at a special case, as studied in [7]. Chung has shown that large bandwidth can be performed using iterated kernel smoothing formula. This eliminates the need for geodesic distance calculation. Nonetheless, if the underlying function is very smooth, large number of iterations is necessary, and as a result the computation would be less efficient than starting initially with larger bandwidth. In the following section we give performance comparisons between the kernel constructed using only 1st-order neighbors and our kernel, which is constructed using up to 3rd-order neighbors, on cortical thickness measurements.

3.2 Geodesic distance calculation

As mentioned previously, if only traveling along edges is allowed, then the problem of finding minimal path between points on a polygonal surface is equivalent to a discrete optimization problem on graph. We now solve the problem using dynamic programming.

In DP set up, our graph is Ω , having finitely many nodes. The nodes in the space are all the vertices $v_k \in \Omega$. Each pair of nodes has an associated cost $C(i,j)$, which is defined to be the length of the edge connecting vertices v_i and v_j , given in millimeters. If v_i and v_j have no connecting edges (v_i and v_j not the 1st-order neighbors of each other) then infinite cost, $C(i,j) = \infty$, is assigned. Let L be the total number of vertices in Ω . To preserve our previous notations, nodes are renamed and re-indexed to be $\{n_1, n_2, \dots, n_d\}$, where n_d denotes the destination node. If the objective is to calculate the heat kernel at v_f on the cortical surface, then $n_d = v_f$ and $\{n_1, n_2, \dots, n_{L-1}\} = \{N_1(f) \cup N_2(f) \cup N_3(f)\}$.

Since the costs are nonnegative, it is clear that the number of hops in the optimal path can be at most L . The problem is formulated such that theoretically we always take L move from n_i to n_d , $i = 1, \dots, L-1$. In actuality this corresponds to the possibilities of degenerate moves, i.e., staying at the same node, since the situation of the optimal path passing every node in the space rarely happens.

For $i = 1, \dots, L$ and for $k = 1, \dots, L-1$, define [4]:

$J_{L-1}(i)$ = optimal cost for getting from i to d in one move.

$J_k(i)$ = optimal cost for getting from i to d in $(L-k)$ moves.

Then, the optimal cost for getting from i to d is $J_0(i)$. The dynamic programming equation (DPE) is given by

$$J_k(i) = \min_{j=1, \dots, L} \{C(i, j) + J_{k+1}(j)\} \quad ; k = 0, 1, \dots, L-2$$

and

$$J_{L-1}(i) = C(i, d) \quad ; i = 1, \dots, L$$

Technical details of Matlab implementation for this DPE can be found in appendix A.

Now let's qualitatively explain the algorithm above. The basic approaches are "thinking backward" or "cost-to-go". We begin by noting that $J_{L-1}(i)$ are readily available, and only $n_i \in N_1(d)$ would have finite costs. Next, the first sub-problem is to find $J_{L-2}(i)$ (taking only two hops from n_i to n_d), which involve new calculations for the costs of moving from $n_i \in \{N_1(d) \cup N_2(d)\}$ to any nodes in $N_1(j)$ in one hop. Then those are

combined with $J_{L-1}(i)$ for the resulting optimal cost of this sub-problem. Proceed with $J_{L-3}(i), J_{L-4}(i) \dots$, until $J_0(i)$, which is a final solution for the original problem, is reached. The solutions $J_0(i)$ for $i = 1, \dots, L$ gives the kernel weights $d(v_i, v_f)$ for $v_i \in \{N_1(f) \cup N_2(f) \cup N_3(f)\}$. The process is repeated for all $v_k \in S_V$ to complete the construction of the heat kernel for the entire surface.

3.3 Kernel weights computation

Using the results from 3.1 and 3.2, we rewrite the formula for the Gaussian kernels on the cortical surface as,

$$\tilde{K}_\sigma(v_j) = \frac{K_\sigma(v_j) \mathbf{1}_{\Omega_i}(v_j)}{\sum_{\forall v_j \in \Omega_i} K_\sigma(v_j)} ; \quad i = 1, \dots, V \quad (11)$$

$$\text{where } K_\sigma(v_j) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{d^2(v_i, v_j)}{2\sigma^2}\right); \quad \forall v_j \in \Omega_i.$$

The only parameter left to be defined in $K_\sigma(v_j)$ is the bandwidth σ . It needs to be sufficiently small in order for the weights of the kernel to be negligible outside Ω_i . However, if σ is too small, the weights would diminish for $\{v_k\} \in N_3(j)$, and we would not utilize the entire constraint region. In our kernel construction, the minimum distance from the center vertex v_i to its farthest neighbor within Ω_i is found to be 6.32 mm, $i = 1, \dots, V$. We use the fact that approximately 99% of the area under a Gaussian curve, $N(0, \sigma^2)$, falls within $\pm 3\sigma$ to constrain the bandwidth selection so that $3\sigma \leq 6.32$. We pick $\sigma = 2$, which ensure that we utilize the entire region of Ω_i and at the same time still satisfy the first constraint that the tail region be negligible outside Ω_i . Also note that since 6.32 is the minimum of all distances, there would be some Ω_i that the kernel weights decay at a faster rate, which imply that σ may be slightly too small. This consequence can be avoided by applying iterated kernel smoothing.

After σ is selected, the final step is to substitute the values of σ and $d(v_i, v_j)$ into equation (11) to finally compute the kernel weights.

4 Experimental results

After the kernels have been constructed, we apply the Gaussian kernel smoothing to cortical thickness measurements. The data is assumed to follow the additive model:

$$Y(x) = \mu(x) + \varepsilon(x)$$

where $\mu(x)$ is the true cortical thickness and $\varepsilon(x)$ is a zero-mean Gaussian random field.

The smooth data is $\hat{\mu}(v_i) = \tilde{K}_\sigma(v_i) * Y(v_i)$, $i = 1, \dots, V$.

The original thickness measurements and data histogram are shown in figure 6.

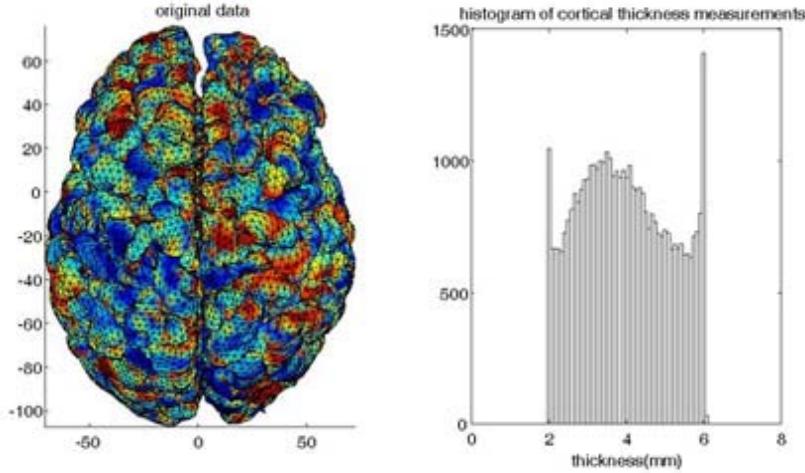


Figure 6: Right column: Original thickness measurement overlayed on the cortical surface. Left column: Data histogram.

The cortical surface used in the experiment has 81920 faces and 40962 vertices. The measurements are smoothed using two different kernels. First, we use our geodesic-based heat kernel, which from now we termed *geodesic kernel* for convenience. The second kernel is what we call the *1st-order heat kernel*—a heat kernel derived using only 1st-order neighbor information, as explained in [7]. The results from the two methods are compared.

The left column of figure 7 shows the data after 4 iterations of smoothing using geodesic kernel with bandwidth $\sigma = 2$, where it gives a decent amount of smoothing. Next we would like to compare this result with the result from the 1st-order heat kernel. In order to obtain the appropriate number of iterations for the 1st-order heat kernel that is comparable in performance with the geodesic kernel result in some sense, we use the geodesic kernel result as our reference and plot the total squared error between the two smoothed data sets as a function of iterations, i.e.,

Let $D_{ref}(i)$ be the data value at vertex v_i , obtained after 4 iterations of geodesic kernel smoothing with $\sigma = 2$.

Let $D_p(i)$ be the data value at vertex v_i , obtained after p iterations of 1st-order heat kernel smoothing with $\sigma = 1$.

Then,

$$p_{optimal} = \min_p \left\{ \frac{1}{V} \sum_{i=1}^V (D_{ref}(i) - D_p(i))^2 \right\}. \quad (12)$$

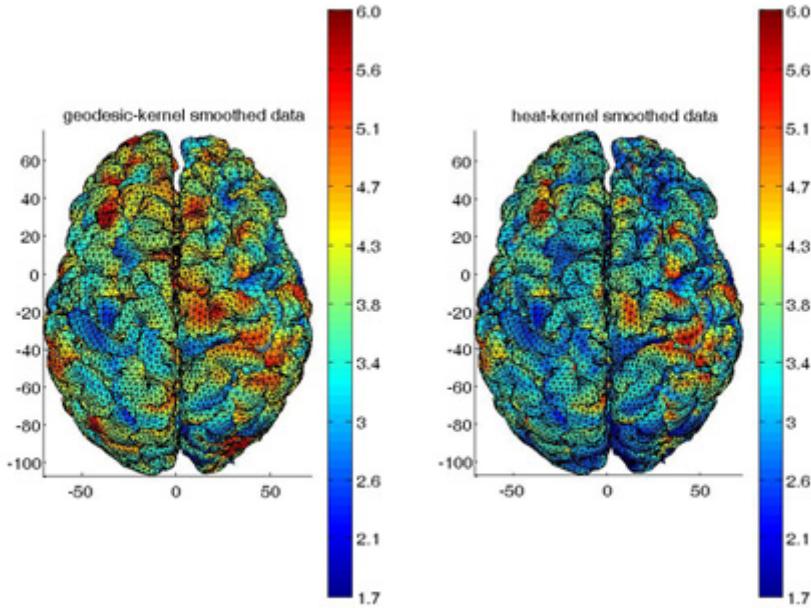


Figure 7: Comparison of smoothing results obtained using geodesic kernel (left) and 1st-order heat kernel (right)

The plot between the error, as given in the bracket of equation (12), and the number of iteration p is shown in figure 8. From this plot we found the optimal number of iteration to be 176. We then perform 176 iterations of 1st-order heat kernel smoothing on cortical thickness measurement; the resulting smoothed data is shown in the right column of figure 7.

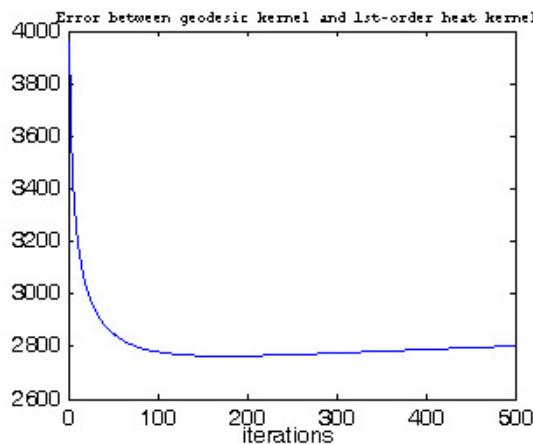


Figure 8: Errors between geodesic kernel result and 1st-order heat kernel result

It takes 1.8750 seconds to calculate all kernel weights and 2.5150 seconds for 4 iterations of geodesic kernel smoothing with $\sigma = 2$. For 1st-order heat kernel smoothing, kernel weight calculation takes 1.1090 seconds and 176 iterations of smoothing takes 16.2040 seconds.

The histograms of the results from the two kernel smoothing methods are also shown (see figure 9). It is evident from the histograms that the smoothed data using geodesic kernel is more Gaussian. It is also observed in figure 7 that the results between the two kernel methods are not really similar pointwise. After trial-and-error, the result of 1st-order heat kernel after 90 iterations gives the most visually equivalent result to the geodesic kernel smoothing, as illustrated in figure 10.

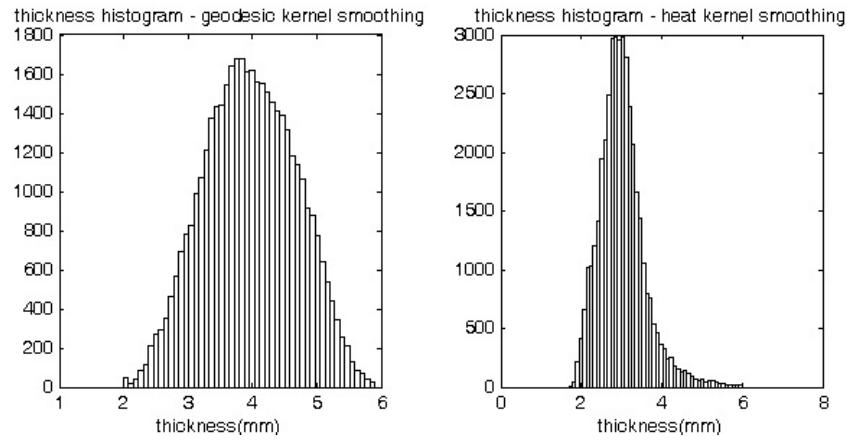


Figure 9: Histograms of smoothed cortical thickness from geodesic kernel smoothing and 1st-order heat kernel smoothing

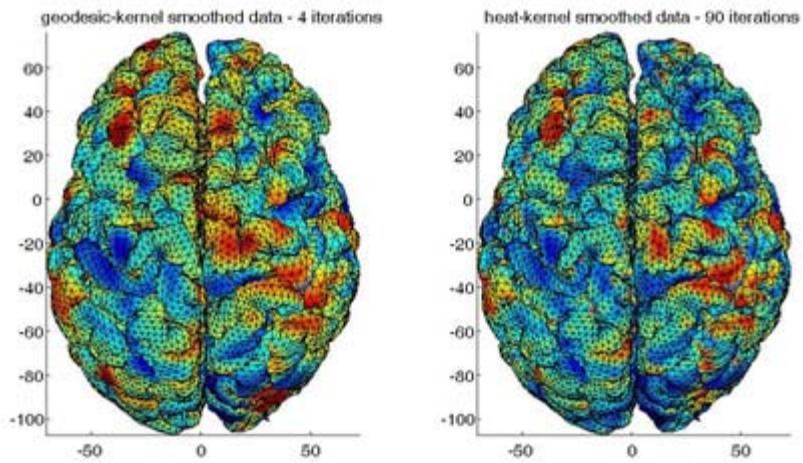


Figure 10: Visually equivalent smoothing with geodesic kernel (left) and 1st-order heat kernel (right)

5 Discussions

The only computationally demanding part of our geodesic-based heat kernel construction is the data pre-processing step, where current implementation required the neighbor information to be stored in a specific format (see Appendix A). The pre-processing step takes about 15-20 minutes per cortical surface. Nonetheless, it is important to note that this pre-processing step has to be run only once per surface. Therefore, for applications in image analysis where chronic diseases or evolutional growth in one subject is monitored over a long period of time, the computational efficiency of the smoothing process would outweigh this pre-processing step. An improved version of this step is still under development.

The experimental results show promising results of geodesic-based heat kernel construction on cortical surface. The first advantage of the geodesic kernel over the 1st-order heat kernel is its computational efficiency. Since geodesic kernel incorporates more data points into each smoothing iteration, a few numbers of iteration are sufficient for the smoothed data to become Gaussian. It is evident that the smoothed data after more than a hundred iterations of 1st-order heat kernel does not have the desired Gaussianness and still have clusters of peaks and valleys instead of a more globally smooth effect, as seen in the result of geodesic kernel. In addition, the construction steps of geodesic kernel are simple, intuitive, and easily extended. Its equivalent formulation to the widely-used Gaussian kernel also suggests the application of this geodesic kernel to any existing applications of Gaussian kernel in image analysis. The possible extension of the work done in this report is the generalization to a case of an anisotropic kernel, as well as more validation results in terms of performance analysis in comparison to other popular choices of kernel.

6 References

1. Baillet, et.al. Electromagnetic Brain Mapping. IEEE Signal Processing Magazine:14-30, November 2001.
2. Barnes, et.al. Quantification of the relationship between MEG and BOLD images of brain function. IEEE workshop on Statistical Signal Processing, October 2003.
3. Bellman, R. Dynamic Programming. Princeton University Press, New Jersey, 1957.
4. Bertsekas, D.P. Dynamic programming: Deterministic and Stochastic Models. Prentice-Hall, New Jersey, 1987.
5. Chen J., and Han, Y. Shortest Paths on a Polyhedron. Proceedings of the 6th annual symposium on Computational geometry: 360-369, 1990.
6. Chung, M.K. STAT992 lecture notes, spring 2004.
7. Chung, M.K and Taylor, J.E. Diffusion Smoothing on Brain Surface via Finite Element Method. 2004 IEEE International Symposium on Biomedical Imaging.
8. Chung, M.K. PhD. Thesis. McGill University, Montreal. 2001.
9. Dale AM, Fischl B, Sereno MI, Cortical surface-based analysis I: Segmentation and surface reconstruction. Neuroimage 1999 Feb;9(2):179-94

10. Dijkstra, E.W. A note on two problems in connection with graphs. *Numer.Math.*, vol.1:269-271, 1959.
11. Lafferty, J. and Lebanon, G. Diffusion Kernels on Statistical Manifolds. 2004 Technical Report, Carnegie Mellon University.
12. Mitchell, J., Mount D., and Papadimitriou, C. The Discrete Geodesic Problem. *SIAM journal on Comp.*, vol.16 (4):647-668, 1987.
13. Ramsay, J.O. and Silverman, B.W. *Functional Data Analysis*. Springer-Verlag, New York, 1997.
14. Rosenberg, Steven. *Lectures on the Laplacian on a Riemannian Manifold*. Keio University, 1993.
15. Sharir, M. and Schorr, A. On Shortest Paths in Polyhedral Spaces. *SIAM journal on Comp.*, vol.15(1):193-215, 1986.
16. Simonoff, J.S. *Smoothing Methods in Statistics*. Springer-Verlag, New York, 1996.
17. Wolfson, E. and Schwartz, E.L. Computing Minimal Distances on Polyhedral Surfaces. *IEEE Trans. on Pattern anal.and Mach. Intel.*, vol. 11(9):1001-1005, 1989.

Appendix A – Matlab implementation of geodesic distance calculation

Let M_1 be the maximum number of 1st-order neighbors for any v_i , $i = 1, \dots, V$.

Let M_3 be the maximum number of neighboring vertices of any vertex, counting the 1st-, 2nd- and 3rd-order neighbors separately and taking the maximum of the three.

The first neighbor information obtained from function `NMIGetmesh` is stored in the $[V \times M_1]$ matrix **sNbr**.

1) This is the data pre-processing step. By calling function `GetNeighbor`, two variables are calculated, namely **Nid** and **Ndist**.

Ndist is a $[V \times M_1]$ matrix; each element $\text{Ndist}(i,j)$ of **Ndist** is the pairwise distance between v_i and its j^{th} 1st-order neighbor, whose index is given by the element **sNbr**(i,j).

Nid is a $[V \times 3 \times M_3]$ 3-dimensional matrix, with the i^{th} $[3 \times M_3]$ sub-matrix stores the neighboring information needed for kernel weights calculation for v_i . The first row is the indices of the 1st-order neighbors of v_i , the second row the 2nd-order neighbors, and the third row the 3rd-order neighbors, respectively.

2) Now we will explain how to find the geodesic distance for EACH vertex v_i .

The function `ShortestPath` is called. This function reads the neighboring information from the i^{th} $[3 \times M_3]$ matrix of **Nid** and finds the corresponding pairwise distances between v_i and all its neighbors. The initial “distance” matrix is constructed as follow.

Note that since M_3 is the maximum number of neighbors, some elements of the $[3 \times M_3]$ matrix would be zeros. Let m be the total number of non-zero elements. v_i and all neighbors are re-indexed as $\{v_1, v_2, \dots, v_{m+1}\}$. Then, construct an $[(m+1) \times (m+1)]$ matrix **S** and initialize **S** with the cost $C(i,j)$, as described in section 3.2.

Remarks: $S(i,i) = 0$, $i = 1, \dots, m+1$, since the distance from one node to itself is zero.

$$S(i,j) = S(j,i).$$

3) the initial distance matrix **S** is passed to the function `DP`. The pseudocode of DP is

```

for i = 1:m+1
    for j = 1:i
        S(i,j) = min {S(i,k) + S(k,j)}
        for all k such that node i connects to node k.
            S(j,i) = S(i,j)
    end
end

```

4) The entries of **S** are updated by the function `DP` to be the shortest distances from v_i to all the neighbors. Then we take the entries of **S** and create the resulting matrix **SP**, which has a one-to-one mapping to **Nid**; each entry of **SP** corresponds to the shortest distance from v_i to its neighbors, whose indices are given by **Nid**. Lastly, **SP** is passed to the function `Kernel` for kernel weights calculation.