Homework 4: Laplacian and Vector Fields

Due April 21, 2021

This is the fourth homework assignment for 6.838. Check the course website for additional materials and the late policy. You may work on assignments in groups, but every student must submit their own write up; please note your collaborators, if any, on your write up. **Submit your code as** `6838-hw4-<yourkerberos>.zip` **and writeup as** `6838-hw4-<yourkerberos>.pdf`**, where** `<yourkerberos>` **is replaced with your MIT Kerberos ID.**

This homework is based on three papers on geometry processing. You are encouraged to read them fully, but the homework will only rely on select parts of them. For simplicity, you can assume the meshes used in this set have no boundary.

1. Geodesics in Heat [GIH]
2. The Vector Heat Method [TVHM]
3. An Operator Approach to Tangent Vector Field Processing [OATVF]

The goal of this problem set is to get used to reasoning about smooth and discrete vector fields on surfaces. None of the coding problems should require more than 20 lines of code. If you find yourself writing a lot more code than that, take a moment to step back and think about the problem at a high level, or come to office hours.

You should familiarize yourself with the code in `utils/`. The `MeshData` structure there contains several useful mesh quantities already. Also note that in our boilerplate code, vector fields will be stored as $N \times 3$ matrices or as $3N \times 1$ vectors. In the latter case, the xyz coordinates of each vector are kept contiguous.

When debugging, it is often useful to ask yourself the following questions. Do the dimensions of your vectors make sense? Is this a vertex-valued function or a face-valued function? Does the function evaluate to a scalar or a vector? Do you need a minus sign in front of your cotangent Laplacian?

Problem 1 (**Helmholtz decomposition** (20 points))**.** The Helmholtz decomposition states that a vector field defined on $\mathbb{R}^3$ can be decomposed in the following way:

$$V = \underbrace{\nabla \zeta}_{\text{curl free}} + \overbrace{\nabla \times W}^{\text{divergence free}} \tag{1}$$

for some scalar field ζ and vector field W.

(a) Given a vector field V, suppose you want to find the portion of it that is curl free i.e. $\nabla\zeta$. Write a linear PDE whose solution obtains ζ. Your answer should not reference W.

(b) Given an arbitrary vector field V, suppose you want to find the portion of that vector field that is divergence free i.e. $\nabla \times W$. Write an equation expressing the divergence free component of V. You can assume you have access to ζ via part (a).

You'll use both of these results later to compute geodesic distances and to simulate an incompressible fluid. Both of these are also frequently-used steps in other geometry processing pipelines.

Problem 2 (Geodesic distance from the Laplacian (40 points)**).** The heat kernel $k_{t,x}(y)$ measures how much heat diffuses to y in time t from an initial heat distribution $u_0 = \delta_x$. Varadhan's formula states that the geodesic distance $\phi(x, y)$ between a pair of points x and y on a Riemannian manifold is related to the heat kernel $k_{t,x}(y)$ in the following way:

$$\phi(x, y) = \lim_{t \to 0} \sqrt{-4t \ \log \ k_{t,x}(y)} \tag{2}$$

One can imagine a naïve algorithm for computing geodesic distances from a source vertex x to a target vertex y by computing heat diffusion for a small t and applying Varadhan's formula. However, this method results in noisy geodesic distance measurements.

(a) What can go wrong in practice if t is too large? What if t is too small?

To obtain a more robust measurement of geodesic distance, GIH [Algorithm 1] proposes several modifications. Using the eikonal equation we know that the gradient of a distance function will always have unit norm. We can leverage this fact by computing the gradient of the small time heat kernel, normalizing it, and then finding the scalar field whose gradient is the normalized gradient field. Going forward, use $t = \sqrt{h}$, where h is the average edge length of the mesh.

(b) In `'geodesicDistance'` use semi-implicit time integration to compute the small time heat kernel. *Note:* the heat equation is $\dot{u} = \Delta u$, meaning you have already implemented heat diffusion in a previous homework!

(c) In the function `getDivGrad`, construct the discrete gradient operator, a sparse $3|F| \times |V|$ matrix which takes a vertex-valued scalar field and outputs a gradient vector per triangle face. These comprise the discrete gradient vector field as specified in lecture 14. In `geodesicDistance` compute the gradient of your heat kernel. Attach a screenshot visualizing the resulting vector field.

(d) Given a single vertex, its adjacent triangles, and a tangent vector per triangle, one can quantify a discrete divergence. In the function `getDivGrad`, construct the discrete divergence operator `Div`, a sparse $|V| \times 3|F|$ operator which computes the vertex-valued divergence given a face-valued vector field following the discretization specified in GIH [Section 3.2.1]. In `geodesicDistance` compute the divergence of your normalized gradient field. Attach a screenshot visualizing the divergence.

Hint: Your operators should satisfy the identity $\nabla \cdot \nabla = -\Delta$, or discretely `Div * Grad = -data.cotLaplacian`.

(e) Implement the last step of GIH [Algorithm 1] to obtain geodesic distances in `geodesicDistance`. Note that this last step is an application of the equivalent of 1(a) on a surface. For the `moomoo.off` mesh and the default provided source vertex, attach a screenshot depicting which vertex is furthest from the source.

Problem 3 (Parallel Transport from the Connection Laplacian (20 points)**).** You have seen the cotangent Laplacian which can be used to diffuse scalar fields on a triangle mesh. Suppose you wanted to diffuse a tangent vector field instead. Naïvely one could apply the cotangent Laplacian three times, once for each xyz-coordinate. This has the unfortunate effect of making the vectors leave their respective tangent spaces. In this problem you will implement the **connection Laplacian** and use it to diffuse vector fields without leaving their tangent spaces, as well as to parallel transport a vector across a triangle mesh.

One can derive the cotangent Laplacian as the Hessian of the Dirichlet energy:

$$D[\phi] = \sum_{ij \in E} \underbrace{(cot\theta_k^{ij} + cot\theta_l^{ji})}_{w_{ij}} (\phi_j - \phi_i)^2 \tag{3}$$

where ϕ is a scalar field specified at vertices of the triangle mesh, and E is the edge set of the mesh. Similarly, one can derive the connection Laplacian as the Hessian of the vector Dirichlet energy:

$$D[V] = \sum_{ij \in E} w_{ij} |V_j - R_{ij} V_i|_2^2 \tag{4}$$

where V is a vector field specified at vertices of the triangle mesh. For this problem we will use vertex normals, which also define a tangent space per vertex. R_{ij} is the smallest rotation taking the normal vector of vertex i to the normal vector of vertex j.

(a) Complete the `buildConnectionLaplacian` function in `vectorDiffusion{.jl,.m}`. (Hint: take a look at the differences between (3) and (4) and think about what modifications that implies for the cotangent Laplacian)

When you diffuse a tangent vector field that is nonzero at only one vertex for time t you obtain the expansion in TVHM [Equation 5]. For small time t, this formula approximates parallel transporting the single vector source along single-source geodesics across the domain.

(b) Run the `vectorDiffusion` script with different values of `dt` and note the effect on the resulting vector field for large and small time steps. Attach screenshots of these diffused fields with labeled dt.

Problem 4 (Operator Approach to Tangent Vector Fields (20 points)**).** In this problem we will interpret a vector field by its action on scalar fields. This will ultimately lead to a basic fluid simulation implementation. In particular, a face-valued vector field V can be represented by the directional derivative operator

$$D_V^F(\phi)_t = V_t \cdot (\nabla \phi)_t \tag{5}$$

where $t \in T$ is a triangle of the mesh, ϕ is a vertex-valued scalar field, and $\nabla\phi$ is the gradient you computed in problem 2. The superscript F means this operator outputs a face-valued scalar

field. This means the D_V^F operator is not square. To make it square, we attach an area weighted averaging operator P that takes a face-valued scalar field and outputs a vertex-valued scalar field:

$$(P\psi)_v = \frac{\sum_{t \in N(v)} \psi_t A_t}{\sum_{t \in N(v)} A_t} \tag{6}$$

where $t \in N(v)$ are triangles adjacent to vertex v, ψ_t is a face-valued scalar field, and A_t is the area of triangle t. Putting these two together gives us the square directional derivative operator

$$D_V = P \circ D_V^F. \tag{7}$$

(a) In `fluidSim{.jl,.m}`, construct the `Dv` operator. This is a sparse $|V| \times |V|$ matrix. *Hint*: the operator P is stored in `data.FtoV`. You may also use your `Grad` matrix from Problem 1.

(b) Implement advection of the color field `color`. Advection can be implemented as follows. Let T_V^t be the operator that advects a scalar field f by a velocity field V for a time t. From OATVF [Lemma 2.5], we have the relation

$$T_V^t f = \exp(tD_V) f. \tag{8}$$

When you run `fluidSim`, you should see the color field circulate around the sphere. Attach a video or screenshots of this advection.

Extra Credit: We now have all the components required to implement a basic incompressible inviscid fluid simulation on triangle meshes. Such a simulation is modeled by the incompressible Navier-Stokes equation

$$\begin{aligned} &\frac{\partial \vec{u}}{\partial t} + \underbrace{(\vec{u} \cdot \nabla)\vec{u}}_{\text{advection}} = -\overbrace{\nabla p}^{\text{incompressibility}} + \underbrace{\vec{f}}_{\text{driving force}} \\ &\nabla \cdot \vec{u} = 0 \end{aligned} \tag{9}$$

We will discretize the fluid velocities with a face-valued tangent vector field $\vec{u}$. Standard algorithms for fluid simulation repeat the following three algorithmic steps:

(c) (**Extra Credit**) Advect (transport) $\vec{u}_i$ along $\vec{u}_i$ for time Δt to obtain $\vec{u}_{i+1}$. Using (8) complete the velocity advection step in `fluidSim`. (Hint: matrix exponentiation can be approximated with just the first three Taylor series terms while remaining sparse.)

(d) (**Extra Credit**) Integrate external driving forces:

$$\vec{u}_{i+2} = \vec{u}_{i+1} + \vec{f}\Delta t \tag{10}$$

(e) (**Extra Credit**: 10 points combined (c)–(e)) Pressure projection

$$\vec{u}_{i+3} = \vec{u}_{i+2} - \nabla p. \tag{11}$$

This removes the divergence component of $\vec{u}_{i+2}$ as we want an incompressible fluid. Use your answers from Problem 1(b) and Problem 2 to complete the pressure projection step in `fluidSim`. Attach screenshots or a short video of your fluid simulation working.