CS 177: Discrete Differential Geometry

Homework 4: Vector Field Design
(due: Tuesday Nov 30th, 11:59pm)

In the last homework you saw how to decompose an existing vector field using DEC. In this homework we'll see that you can apply the same tools to *create* vector fields. Of course, there are many ways one can specify a vector field – for instance, suppose you want to construct a vector field with a specific set of *sources* and *sinks*. Using the code you've already written, you could specify a scalar potential f and take its gradient to get a curl-free vector field $X = \nabla f$. However, it may not be obvious how to construct an f that gives you exactly the features you want. In this homework we'll see how to construct a vector field by simply specifying the locations and types of vector field *singularities*.

1 Discrete Connections

In discrete differential geometry, there are often many ways to discretize a particular smooth object. The hope, however, is that we can discretize all the objects in a given *theory* so that relationships from the smooth theory hold in the discrete picture. For instance, when we looked at Hodge decomposition we discretized the *exterior derivative* (d), *Hodge star* ($\star$) and *1-forms* in such a way that the Hodge decomposition theorem has an *identical* expression in both the smooth and discrete world: $\omega = d\alpha + (-1)^{n(p+1)+1} \star d \star \beta + \gamma$. In particular, we represented vector fields as *discrete 1-forms*, i.e., a number associated with each oriented edge giving the circulation along (or flux through) that edge.

In this homework we're going to look at a different theory: the theory of *connections* and *parallel transport*. As a result, we're going to indirectly represent a vector field through an *angle-valued dual 0-form*. More plainly, we're going to store an angle on each face that defines the local change of direction of the field. Note that this representation ignores *magnitude*, so what we're really working with is a *direction field*. Before going too much further with the discrete theory, though, let's first talk about the objects we want to discretize.

1.1 Parallel Transport

A *surface* M is a space that "locally" looks like the flat Euclidean space $\mathbb{R}^2$. The surface of the Earth is a perfect example: if we "zoom in" far enough, it's hard to tell that we're not walking around on a plane. So, we often cheat and approximate the surface by a plane at each point. This plane is called the *tangent plane* and vectors in this space are called *tangent vectors*. (Of course, now that you've been working with discrete surfaces for a little while this idea should come as no surprise to you – in fact, in the discrete setting we essentially *replace* the smooth surface with a collection of tangent planes, namely, the faces of our mesh!) From here on out we're going to consider *tangent vector fields*, i.e., vector fields consisting of tangent vectors only.
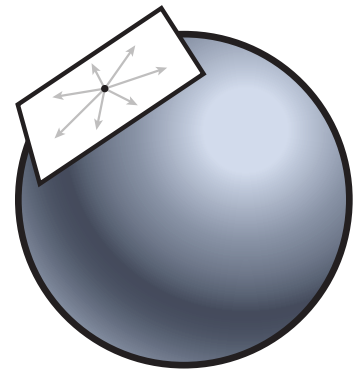
What if we want to move from one point of M to another while preserving the direction of some tangent vector u? Well, if M is completely flat (like the plane itself) then the most natural thing is to slide u from one point to the other along a straight path, keeping the angle with some reference direction fixed. This process is called *parallel transport*, and the vectors u and u' are, as usual, said to be *parallel*. Defining parallel transport on a *curved* surface is a bit trickier: we have many paths to choose from, and worse, we no longer have a consistent reference direction. So unfortunately, the notion of "same direction" is not well-defined.
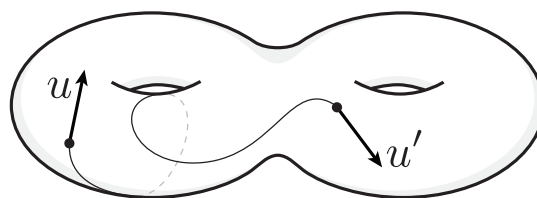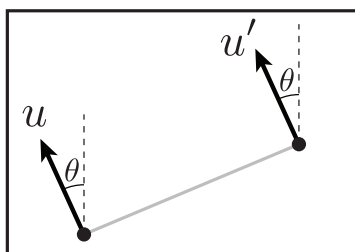
1.2 Connections

Still, having some notion of "same direction" could be very convenient in a lot of situations. So, rather than looking for some "natural" definition, let's define for ourselves what it means to be parallel.

Ok, but how do we do that? One idea is to specify a bunch of maps f that immediately "teleport" vectors from one tangent plane to another. We could then say that – *by definition* – vectors u and v are parallel if $f(u) = v$. Unfortunately we'd have to specify how this map works for *every pair* of tangent planes on our surface, *and* ensure that all these maps agree – that's a lot of work. But we're on the right track.
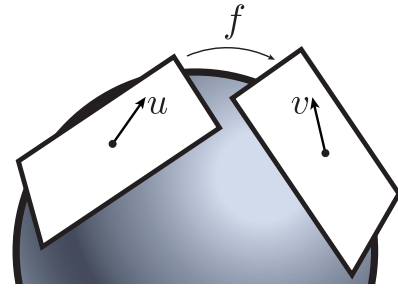
Instead, let's consider only *nearby* tangent planes. In other words, if we move some *small* distance ϵ, which vector v in the new tangent space should be considered parallel to our original vector u? Well, we can provide an answer for this question just as before: by specifying a map f between nearby tangent planes. Moreover, if ϵ is an *infinitesimally* small distance, this information defines a *connection* on $\mathcal{M}$, i.e., a map that "connects" nearby tangent spaces[1]. Of course, once we have a connection we can easily parallel transport vectors along longer paths: just take a bunch of really small steps!

Here's an experiment you can do at home to help make these ideas more "real": Take a stiff piece of cardboard and draw some arrows on it. Now roll it around on the surface of a basketball for a while. In effect, you're defining a map between the tangent plane where you first set down the cardboard and the tangent plane at the current location. And the rolling and twisting motion you apply at any given moment defines a connection (at least along a particular path).

As you roll your cardboard around for a while, you might notice something funny happens. (No, I don't mean the looks you're getting from your friends.) In particular, try rolling the cardboard back to the point where you initially set it down. Are the arrows pointing the same way as they were originally? What if you take a different path along the surface of the ball?

Even after noodling around with the basketball, maps between infinitesimally-close tangent spaces can be hard to think about clearly. Fortunately, everything gets a lot simpler in the discrete case. For one, we have only finitely many tangent planes to deal with (namely, the faces of our mesh). More importantly, for each tangent plane we have only a *very* small number of "nearby" tangent planes (namely, the neighboring faces). So specifying one map f for each neighboring pair is not so bad.

How exactly should we specify one of these maps? Well, for a given a pair of triangles (i, j), we can imagine rigidly unfolding them the plane, translating a vector from one to the other, applying a rotation by some small angle θ_{ij}, and the rigidly "refolding" these triangles into their initial configuration, as illustrated below. Equivalently, if we imagine we have a discrete basketball with *flat* faces, we can imagine "see-sawing" our cardboard across the shared edge so that it comes into contact with the new face, then spinning it slightly around the normal by the same angle θ_{ij}. (Since we care only about *directions* and not magnitudes, we can omit any stretching/shearing motion.)
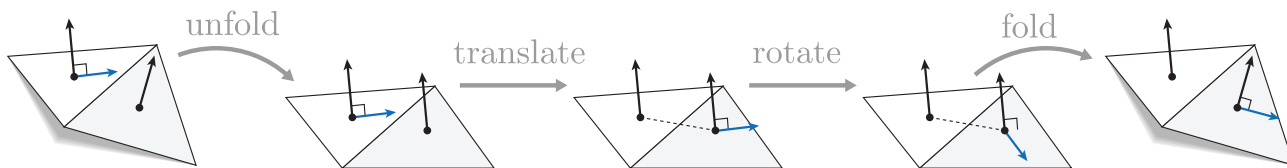
So at the end of the day, we just need to specify a single angle $\varphi_{ij} \in \mathbb{R}$ for each oriented *dual* edge in our mesh. We should also make sure that $\varphi_{ji} = -\varphi_{ij}$ – in other words, the motion we make going from face j to face i should be the opposite of the motion from i to j. Doing so ensures that our notion of "parallel" is consistent no matter which direction we travel. Such a collection φ of angles on dual edges is called a *discrete connection*.

By the way, is this object starting to sound familiar? It should be! In particular, we have a single number per oriented dual edge, which changes sign when we change orientation. In other words, φ is a *dual, discrete 1-form*.

---

[1]Strictly speaking, we're talking about an *affine connection* on the tangent bundle of a surface; in general, connections can get a lot more interesting!

1.3 The Levi-Civita Connection

In terms of the picture above, we effectively said that an angle $\varphi_{ij} = 0$ means "just translate; don't rotate." If we set *all* of our angles to zero, we get a very special connection called the *Levi-Civita* connection[2]. As you can see, the Levi-Civita connection basically tries to "twist" a tangent vector as little as possible as it moves it from one point to the next.

Not surprisingly, the Levi-Civita connection is also well-defined in the smooth setting. There are many ways to formulate the smooth version, but a particularly simple characterization is given by Kobayashi:

> **Kobayashi's Theorem**. *The Levi-Civita connection on a smooth surface is the pullback under the Gauss map of the Levi-Civita connection on the sphere.*
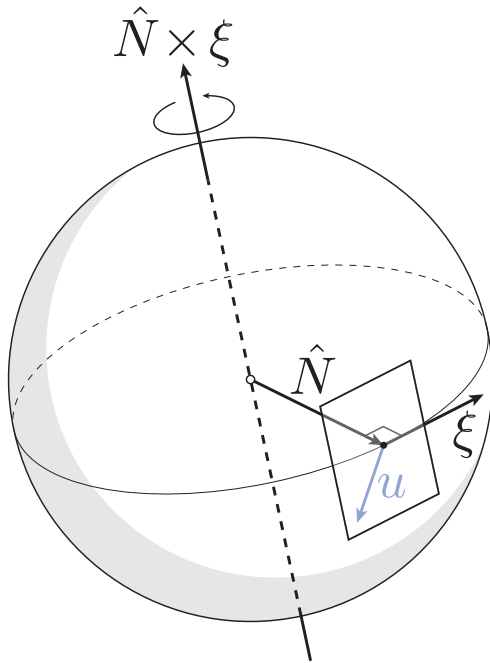
What does this all mean? First, you may remember the Gauss map from Homework 1 (Section 3.3) where we use it to define *angle defect*. In particular, the Gauss map $\hat{N} : \mathcal{M} \to S^2$ takes a point on the surface to its corresponding unit normal, which can be thought of as a point on the unit sphere. (You can just as easily imagine that the Gauss map takes a *tangent plane* and spits out its unit normal.)

Ok, but what's the Levi-Civita connection on the sphere? Well, we said that Levi-Civita tries to "twist" vectors as little as possible. On a sphere, it's not hard to see that the motion of least twist looks like a rotation of the tangent plane "along" a great arc in the direction ξ of parallel transport. More explicitly, we want a rotation around the axis $\hat{N} \times \xi$, (where $\hat{N}$ is the normal of our original tangent plane).

Altogether, then, Kobayashi's theorem says the following. If we start out with a tangent vector u_0 on $\mathcal{M}$ and want to transport it in the direction ξ_0, we should first find the corresponding tangent vectors u and ξ in the tangent plane with normal $\hat{N}$ on the sphere. (Of course, these are just the same vectors!) We should then apply an (infinitesimal) rotation along the great arc in the direction ξ, dragging u along with us.

Exercise 1. Use Kobayashi's theorem to justify the "unfold, translate, refold" procedure that is used to define the discrete Levi-Civita connection.

*(Hint: think about the unfolding as a rotation.)*

1.4 Holonomy

At this point you may be wondering what all this stuff has to do with vector field design. Once we define a connection on our mesh, there's an easy way to construct a vector field: start out with an initial vector, parallel transport it to its neighbors using the connection, and repeat until you've covered the surface. One thing to notice is that the vector field we end up with is completely determined by our choice of connection (see below). In effect, we can design vector fields by instead designing *connections*.

---
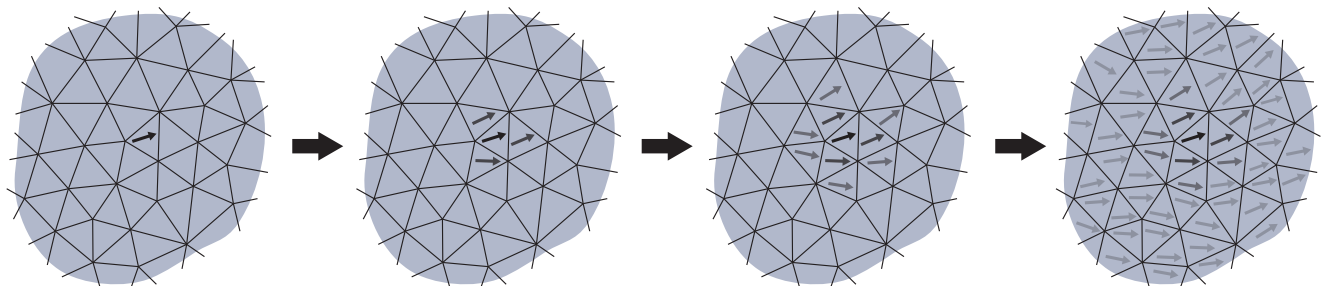
[2]Those of you with some geometry background may note that a discrete connection really encodes the *deviation* from Levi-Civita; it should not be thought of as the connection 1-form itself.

However, something can go wrong here: depending on which connection we use, the procedure above may not give us a consistent definition for a vector field. For instance, consider the planar mesh on the right, and a connection that applies a rotation of 18° as we cross each edge in counter-clockwise order. By the time we get back to the beginning, we've rotated our initial vector ❶ by only $5 \times 18° = 90°$. In other words, our connection tells us that ❶ and ❻ are parallel vectors!
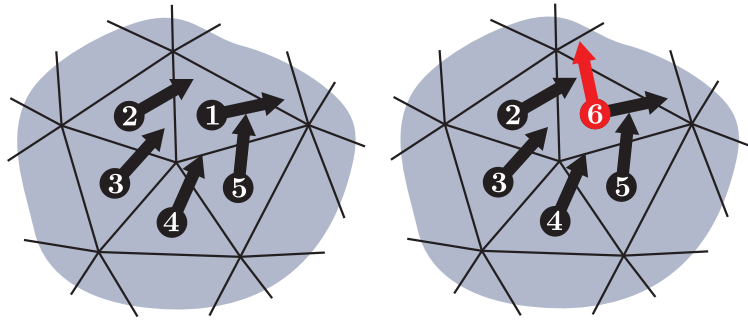
The phenomenon we're experiencing here is referred to as the *holonomy* of the connection. More generally, holonomy is the difference in angle between an initial and final vector that has been parallel transported around a closed loop. (This definition works in both the discrete *and* the smooth setting.)

1.5   Trivial Connections

To have any hope of constructing a consistently-defined vector field, then, it seems that we should at least make sure our connection has *zero* holonomy around every loop. Such a connection is called a *trivial connection*. In fact, the following exercise shows that this condition is sufficient to guarantee consistency everywhere:

Exercise 2. Prove that parallel transport on a surface M is path-independent if and only if the connection on M is trivial.

(Hint: consider two different paths from point a to point b.)

As a result we can forget about the particular path along which a vector is transported, and again imagine that we're simply "teleporting" it from one point to another. In other words, a trivial connection lets us accomplish what we originally set out to do in Section 1.2, *without* specifying a vast amount of information. If we now reconstruct a vector field using a trivial connection we get a *parallel vector field*, i.e., a field where – at least according to the connection – every vector is parallel to every other vector. In a sense, parallel vector fields on surfaces are a generalization of *constant* vector fields in the plane. But actually, the following exercise shows that *any* vector field can be considered parallel, as long as we choose the right connection:

Exercise 3. Show that every discrete vector field is parallel with respect to some trivial discrete connection.

(Hint: think about the difference between vector directions in adjacent triangles.)

1.6   Curvature

Ok, so we can use a trivial connection to define a vector field, but how do we come up with a trivial connection? The first thing you might try is using the Levi-Civita connection (after all, it has a nice, straightforward definition). Unfortunately, the next exercise shows that Levi-Civita is not (in general) trivial.

Exercise 4. Show that the holonomy of the discrete Levi-Civita connection around the boundary of any dual cell equals the angle defect of the enclosed vertex.

So unless our mesh is completely flat, Levi-Civita will exhibit some non-zero amount of holonomy. Actually, you may remember (from Homework 1) that angle defect is used to define a discrete notion of *Gaussian curvature*. We can also use a connection to determine curvature – in particular, the *curvature of a connection* (smooth or discrete) over a disk-shaped region $D \subset M$ is given by the holonomy around the region boundary ∂D.
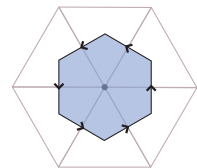
Exercise 5. Show that these two notions of curvature are the same, i.e., show that the curvature of the discrete Levi-Civita connection over any disk D equals the total discrete Gaussian curvature over that region.

(Hint: use induction on the faces in the region.)

(Once again, the same relationship holds in the smooth case – but it is not quite so easy to prove!)

Curvature gives us one tool to test whether a connection is trivial. In particular, a trivial connection *must* have zero curvature everywhere. For this reason we'd be justified in saying that every trivial connection is "flat." But is every flat connection also trivial?
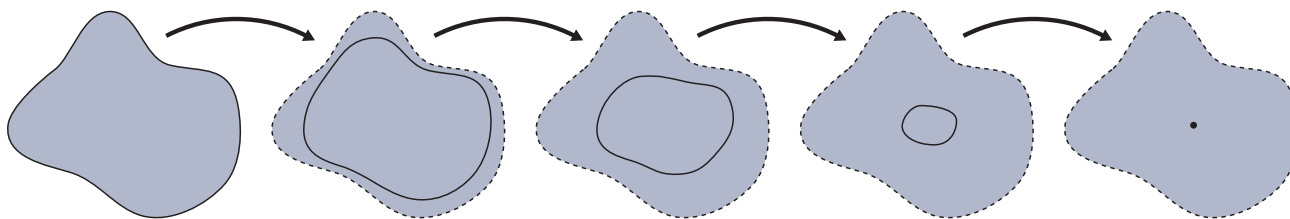
Well, remember that the curvature of a connection is defined in terms of the holonomy around region boundaries. Any such boundary is called a *contractible loop* because it can be continuously deformed to a point without "catching" on anything. But in general, there may be *noncontractible* loops on a surface that *cannot* be described as the boundary of any disk. (For instance, consider the blue loop γ pictured on the torus to the left. Is it the boundary of a disk?)
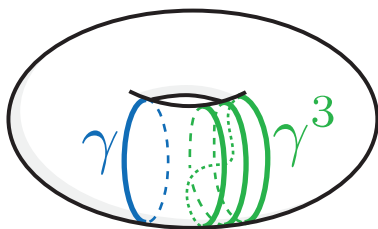
In general a surface of genus g will have $2g$ "basic types" of noncontractible loops called *homology generators*. More precisely, two loops are said to be *homotopic* if we can get from one to the other by simply sliding it along the surface without breaking it at any point. No two distinct generators are homotopic to each other, and what's more, we can connect multiple copies of the generators to *"generate"* any noncontractible loop on the surface. For instance, consider the green loop γ^3, which consists of three copies of γ joined end-to-end.

So, in general, if we want to check if a connection is trivial, we need to know that it has nontrivial holonomy around both contractible *and* noncontractible loops. Or, equivalently, that it has zero *curvature* and nontrivial holonomy around noncontractible loops. As you're about to demonstrate, though, we don't need to check all of the noncontractible loops – just some collection of $2g$ generators.

Exercise 6. Show that we can determine the holonomy around any loop whatsoever as long as we know the curvature at each vertex and the holonomy around a collection of $2g$ generators.

1.7 Singularities

There's one more issue we run into when trying to find a trivial connection. You may remember the *Gauss-Bonnet* theorem from Homework 1, which says that $\sum_{v \in V} d(v) = 2\pi\chi$, i.e., the total Gaussian curvature over a surface equals 2π times the Euler characteristic χ. In fact, this theorem still holds if we replace the Gaussian curvature with the curvature of *any* connection (i.e., not just Levi-Civita).

But there's a problem: didn't we say that a trivial connection should have zero holonomy – hence zero curvature? So unless $\chi = 0$ (i.e., M is a torus) we have a problem!

Fortunately the solution is quite simple: we can allow our connection to have holonomy (hence curvature) around some loops, so long as this holonomy is always a multiple of 2π. As a result, a vector parallel transported around any loop still ends up back where it started, even if it "spins around" some number of times along the way. In particular, we need the curvature at every vertex to equal $2k\pi$ for some integer k, so that the loops around each dual cell have no apparent holonomy. Any vertex where $k \neq 0$ is called a *singularity* because of the effect this additional curvature has on the appearance of the resulting parallel vector field (see below for some examples). As we'll see in the moment, singularities will actually make it *easier* to design vector fields with the desired appearance.
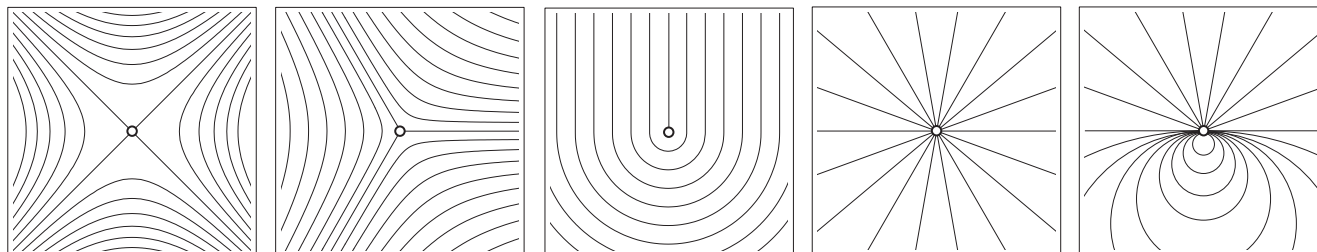
2 Designing Vector Fields

Now on to the fun part: designing vector fields. Believe it or not, you've already written a lot of the code you'll need in Homework 3. But let's take a look at the specifics. To keep things simple we're going to assume that $\mathcal{M}$ is a topological sphere, so you can forget about non-contractible loops for now.

2.1 Computing Trivial Connections

We know that we want to solve for a 1-form φ such that the holonomy around every loop is zero. When φ equals zero on every edge, we know (from Exercise 4) that holonomy equals Gaussian curvature, which we'll denote by the vector $K \in \mathbb{R}^{|V|}$. So, we need to instead find angles φ_{ij} such that

$$d_0^T \varphi = -K, \tag{1}$$

i.e., the holonomy around the boundary of every dual cell should be the *opposite* of Gaussian curvature. (Note that d_0^T gives us the discrete exterior derivative on *dual* 1-forms. Why is this true?) This way the little rotations we apply across each edge exactly "cancel out" the angle defects we'd see if we didn't apply any rotation at all.

We also need to incorporate singularities into this system. That's easy enough: we can still ask that our angles φ_{ij} cancel the usual Gaussian curvature, but they should also yield the prescribed curvature at singularities:

$$d_0^T \varphi = -K + 2\pi k. \tag{2}$$

Here $k \in \mathbb{Z}^{|V|}$ is a vector of *integers* encoding the amount of curvature we want at each vertex. So to "design" a vector field with certain singularities, we just need to set the appropriate entries of k. Of course, we still need to make sure that $\sum_i k_i = \chi$ so that we do not violate the Gauss-Bonnet theorem.

Coding 1. Write a MATLAB routine to compute per-vertex Gaussian curvature following the template below. (Note that this is *not* the same as the *mean* curvature you computed in Homework 2)

```
  function K = get_gaussian_curvature( V, F )
% function K = get_gaussian_curvature( V, F )
%
% Computes the standard discretization of Gaussian curvature at
% mesh vertices, i.e., 2pi minus the sum of the angles around each
% vertex.
%
% INPUT:
%    V - 3x|V| matrix of vertex positions
%    F - |F|x3 matrix of faces as 1-based indices into vertex list
%
% OUTPUT:
%    K - |V|x1 vector of per-vertex Gaussian curvatures
%
```

We now have a nice linear system whose solution gives us a trivial connection with a prescribed set of singularities. One last question, though: is the solution unique? Well, our connection is determined by one angle per edge, and we have one equation to satisfy per dual cell – or equivalently, one per vertex. But since we have roughly three times as many edges as vertices (which you should be able to work out using the Euler-Poincaré formula!), this system is probably *underdetermined*. In other words, there are *many* different trivial connections on our surface.

Which one should we use for vector field design? While there's no "right answer" to this question, it may make sense to look for the trivial connection that's *closest* to Levi-Civita. Why? Well, remember that Levi-Civita "twists" vectors as little as possible, so we're effectively asking for the *smoothest* trivial connection. Computationally, this amounts to looking for the solution to Equation 2 with minimum *norm*, since our angles ϕ_{ij} already encode the deviation from Levi-Civita. As a result, we end up with the following optimization problem

$$\underset{\varphi \in \mathbb{R}^{|E|}}{\text{minimize}} \quad \|\varphi\|$$

$$\text{subject to} \quad d_0^T \varphi = -K + 2\pi k. \tag{3}$$

One way to solve this problem would be to use a gradient-descent type algorithm (like you did for mean curvature flow in Homework 2). However, we can be a bit more clever here by recognizing that Equation 3 is equivalent to looking for the solution to Equation 2 that has no component in the null space of d_0^T. (Any other solution will have larger norm.)

Exercise 7. Show that the null space of d_0^T is spanned by the columns of d_1^T.

(Hint: what happens when you apply d twice?)

Hence, the final algorithm is:

1. Compute *any* solution $\tilde{\varphi}$ to Equation 2 (with the *backslash* operator in MATLAB, say).

2. Project out the nullspace component of $\tilde{\varphi}$ by computing $\varphi = \tilde{\varphi} - d_1^T (d_1 d_1^T)^{-1} d_1 \tilde{\varphi}$.

Coding 2. Write a MATLAB routine that computes a trivial connection on a mesh with spherical topology following the template below. You should *not* explicitly build the inverse matrix $(d_1 d_1^T)^{-1}$ (this will be too slow!), but instead use the backslash operator to solve a linear system for an intermediate variable.

```
  function x = compute_trivial_connection( d0, d1, K, S )
% function x = compute_trivial_connection( d0, d1, K, S )
%
% Computes a set of connection angles x associated with each dual edge of a mesh that
% describe a trivial connection with the specified singularities.  These angles are
% expressed relative to the discrete Levi-Civita connection, i.e., ``translation''
% across the shared edge between two faces.
%
% INPUT:
%    d0 - sparse |E|x|V| matrix representing discrete exterior derivative on 0-simplices
%    d1 - sparse |F|x|E| matrix representing discrete exterior derivative on 1-simplices
%    K  - |V|x1 vector of per-vertex Gaussian curvatures
%    S  - Nx2 vector specifying N singularities as (vertex ID, index) pairs --
%         indices must sum to 2
%
% OUTPUT:
%    x - |E|x1 vector of angles encoding deviations from the Levi-Civita connection
%
```

Coding 3. Write a MATLAB routine that constructs a vector field parallel with respect to a given connection. The basic idea for this routine is discussed in Section 1.4. You may find it useful to create some sort of list for keeping track of triangles that have already been visited. (Feel free to pick the first vector arbitrarily.)

```
  function X = extract_direction_field( V, F, d0, d1, x )
% function X = extract_direction_field( V, F, d0, d1, x )
%
% INPUT:
%    V  - 3x|V| matrix of vertex positions
%    F  - |F|x3 matrix of faces as 1-based indices into vertex list
%    d0 - sparse |E|x|V| matrix representing discrete exterior derivative on 0-simplices
%    d1 - sparse |F|x|E| matrix representing discrete exterior derivative on 1-simplices
%
% OUTPUT:
%    X - 3x|F| matrix specifying a unit vector in each face
%
```

Coding 4. Using the routines from the previous coding exercises along with the provided MATLAB files, execute the following script which computes and displays a vector field with the prescribed singularities. (This script can also be found in the file `vector_field_design.m`.) Initially you should get a vector field that looks like one pictured below. Please hand in an image of your results, plus images of two other vector fields with different types and numbers of singularities. You should also hand in an image of the results you get when you try to reconstruct a vector field using Levi-Civita (you should see several discontinuities in the resulting field).
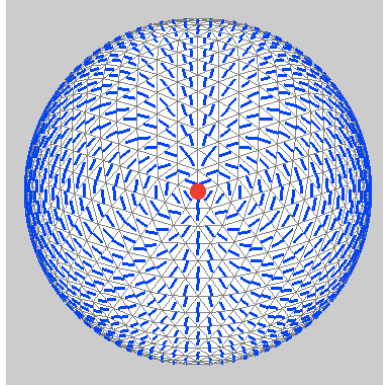

```
%==== INPUT ====================================================================

input_filename = 'test.obj';

% list of singularities as (vertex ID, index) pairs
% NOTE: indices *must* sum to 2!
% ---------------------------------------------------
S = [ 1, 2 ];                          % one singularity of index +2


%==== MAIN PROGRAM =============================================================

stdout = 1;

fprintf( stdout, 'Reading mesh...\n' );
[V,F] = read_mesh( input_filename );

fprintf( stdout, 'Building exterior derivatives...\n' );
[d0,d1] = build_exterior_derivatives( V, F );

fprintf( stdout, 'Getting Gaussian curvature...\n' );
K = get_gaussian_curvature( V, F );

fprintf( stdout, 'Computing trivial connection...\n' );
x = compute_trivial_connection( d0, d1, K, S );

fprintf( stdout, 'Extracting parallel direction field...\n' );
X = extract_direction_field( V, F, d0, d1, x );

fprintf( stdout, 'Displaying result...\n' );
plot_direction_field( V, F, X, S );

fprintf( stdout, 'Done.\n' );
```

A Resources

While implementing your code, you may find it very helpful to consult the following resources:

- Crane, Desbrun, Schröder, *Trivial Connections on Discrete Surfaces*, Computer Graphics Forum 2010. (Available from http://www.multires.caltech.edu/pubs/Connections.pdf)

- *Trivial Connections on Discrete Surfaces* – Conference Presentation. (Available from http://bit.ly/9GxXdg.)