INVSPEC Documentation

Feng Ling

November, 2016

1 Contents

This repository (`https://github.com/levincoolxyz/invspec`) contains a MATLAB implementation of various codes used and results obtained in attempting to tackle the 2D discrete inverse spectral problem. Things are organized in several folders as follows:

1. "/" (root)

   treat this as a typical /src folder (it did not have further structure for now due a lack of interest in managing paths variable in MATLAB, sorry).

2. "/SH"

   this folder contains results of the inverse problem using the spherical harmonic bases. the subfolder named "/SH/forward" contains that of the forward problem.

3. `$MESH`="/../meshes"

   this is where you should populate all the `*.obj` or `*.mat` mesh files. Currently there are a family of bunny meshes, a family of spot the cow meshes (`cow*.mat` are melted cows), and a family of blob meshes.

4. "/mcf"

   in this folder lies the results of (conformal) modified Mean Curvature Flow meshes.

5. "/bunny"

   this folder contains results of the inverse problem on bunny mesh using the simplicial bases

6. "/spot"

   this folder contains results of the inverse problem on spot the cow mesh using the simplicial bases

7. "/harmonic"

   this folder contains results of the inverse problem on some harmonic (low order spherical harmonic symmetric) mesh using the simplicial bases

2 How to Use this

Since everything is in MATLAB, just fire it up and have fun! But seriously, the main "executables" are the script files with name `test_script*.m`. There are also other script files that each purposed to test submodules of the code base. Their usage are summarized below individually.

2.1 gradient verification utility

`gradverify*.m` These scripts verify if the cost functions (both simplicial basis and spherical harmonic basis) return gradients that match finite difference calculations.

2.2 simplicial basis inverse problem

`test_script.m` This is the top level script that attempts to solve the inverse spectral problem in simplicial/linear basis.

One can modify the maximum iterations and error tolerances to their liking and the key parameters specific to our problem are

1. `numeig`, number of dicrete eigenvalues used for the optimization. It indicates a ratio if it is between 0 and 1. If it is larger than 1 it means the exact *integer* number of eigenvalues considered. If it is below 0 it means use all possible discrete eigenvalues;

2. `reg`, a scalar weighting the importance between the cost of matching the spectrum and bilaplacian regularization term ($J = J_{spec} + reg \cdot J_{bi\nabla}$);

3. `method`, BFGS means using the MATLAB built-in fminunc, GD means using the in-house gradient descent with line search (slower but more user control). The gradient descent and line search have been debugged extensively so it should be relatively trustworthy.

4. `input_data`, a structure where the property field `num` (integer) indicates the type of input data and `dat` (string) encodes the data/where to get it.

   (a) `.num=1`, import a mesh in face-vertex format from `*.obj` file named by the `.dat` field.

   (b) `.num=2`, create a spherical mesh of with the total number of vertices equal to the integer encoded by `.dat` field. This uses some mesh optimization written by other people.

   (c) `.num=3`, import a mesh in face-vertex format from `*.mat` file named by the `.dat` field. This file should contain a variable v for vertices and f for face indices.

   (d) `.num=4`, use the steady state Modified conformal Mean Curvature Flow of the target mesh as the input mesh.

5. `target_data`, a structure where the property field `num` (integer) indicates the type of input data and `data` (string or inline function) encodes the data/where to get it.

   (a) `.num=1`, create a mesh based on the input with a small random conformal factor per vertex.

   (b) `.num=2`, create a mesh with a prescribed perturbation from the input mesh along its vertex normals proportional (`pert`) to a scalar function (`.dat` encodes the inline function).

   (c) `.num=3`, import a mesh in face-vertex format from `*.obj` file named by the `.dat` field.

(d) `.num=4`, import a mesh in face-vertex format from `*.mat` file named by the `.data` field. This file should contain a variable v_T for vertices and f_T for face indices.

(e) `.num=5`, specify explicitly the target spectrum in the `.dat` field.

6. `refctl`, a variable (thresholds) that is used by the incomplete adaptive refinement routine. Advise to *ignore and develop your own better version.*

2.3 computation of 3j symbols

`precompute.m`

This script computes the Wigner 3j symbols and their derivatives w.r.t. the spherical harmonic basis conformal coefficients for the *real* spherical harmonics functions (scaled to be on a unit sphere). The only parameter is `maxL`, which indicates the largest degree of spherical harmonics that are being considered. It might be worthwhile to adapt other people's more efficient/accurate computation and storage schemes. e.g. WIGXJPF `http://fy.chalmers.se/subatom/wigxjpf/`.

2.4 spherical harmonic basis forward problem

`SphericalHarmonicTest.m`

This script verifies that our spherical harmonic representation provides good enough conformal factor and spectrum convergence with the simplicial basis, which is known to be correct (in the fine mesh limit) for a variety of reasons (DEC, degree 1 FEM ...).

The only inputs are (1) the string variable `target` that reference the mesh, assuming the path is relative to `$MESH`, and (2) the integer variable `Lrange`, which is the range of different degrees of spherical harmonics that wanted to be explored (to test convergence).

2.5 spherical harmonic basis inverse problem

`test_scriptSH.m` This is the top level script that attempts to solve the inverse spectral problem in the spherical harmonics basis.

One can modify the maximum iterations and error tolerances to their liking and the key parameters specific to our problem are

1. `numeig`, number of dicrete eigenvalues used for the optimization. It indicates a ratio if it is between 0 and 1. If it is larger than 1 it means the exact *integer* number of eigenvalues considered. If it is below 0 it means use all possible discrete eigenvalues;

2. `maxL`, maximal degree of spherical harmonics considered, this impacts execution speed significantly. Advise to use `maxL` in `precomputeRSHI.m` for maximum accuracy.;

3. `numa`, number of spherical harmonic conformal coefficients estimated. So far it seems a number close to `numeig` yields reasonable results, but more exploration is needed as it is not obvious how this interacts with the quality of the solution;

4. `reg`, a scalar weighting the importance between the cost of matching the spectrum and bi-laplacian regularization term ($J = J_{spec} + reg \cdot J_{bi\nabla}$);

5. `method`, BFGS means using the MATLAB built-in fminunc, GD means using the in-house gradient descent with line search (slower but more user control). The gradient descent and line search have been debugged extensively so it should be relatively trustworthy;

6. `input_data`, a structure where the property field `num` (integer) indicates the type of input data and `dat` (string) encodes the data/where to get it. This structure has functions exactly the same as that of 2.2(4);

7. `target_data`, a structure where the property field `num` (integer) indicates the type of input data and `data` (string or inline function) encodes the data/where to get it. This structure has functions exactly the same as that of 2.2(5).

2.6 conformal factor alignments

scalar scaling should be resolved by the opimization, so all that is left are conformal invariances and isometries. The only nontrivial non-isometric conformal invariance that applies to a surface is the double spherical inversions. We can try balancing the factors via M obius balancing. A routine modeled after Prof. Keenan's write-up is implemented unsuccessfully in `Mobiusbalancing.m`. Some debugging tests can be found in the final code block named "mobius balancing test" of `balancingtest.m`

For the left-over rotational SO(3) isometries, there are two ways:

1. naive method using a uniformly random sample of SU2/quaternions plus `fminsearch`

   immediately after running `test_script*.m`, run `rot_test.m` (the key is the have the correct string for the *endname* variable).

2. PCA (extrinsic or intrinsic)

   Personally I think this does not behave nicely enough for noisy results (since it essentially only minimizes the maximum inertia point and minimum inertia point), but YMMV i.e. I might have made a mistake.