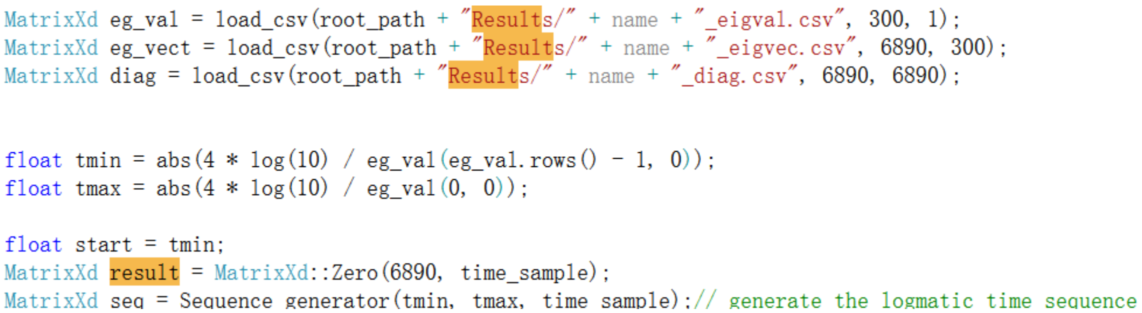
HW5 Question 3

Xu Yin 20205445

December 12, 2021

1 HKS calculation (see Fig. 1)

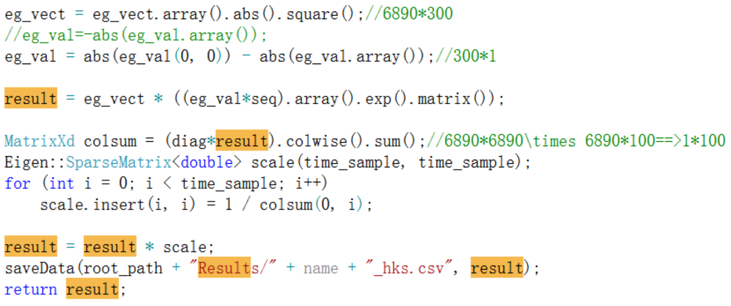
- Generate 100 time samples.



```
MatrixXd eg_val = load_csv(root_path + "Results/" + name + "_eigval.csv", 300, 1);
MatrixXd eg_vect = load_csv(root_path + "Results/" + name + "_eigvec.csv", 6890, 300);
MatrixXd diag = load_csv(root_path + "Results/" + name + "_diag.csv", 6890, 6890);


float tmin = abs(4 * log(10) / eg_val(eg_val.rows() - 1, 0));
float tmax = abs(4 * log(10) / eg_val(0, 0));

float start = tmin;
MatrixXd result = MatrixXd::Zero(6890, time_sample);
MatrixXd seq = Sequence_generator(tmin, tmax, time_sample);// generate the logmatic time sequence
```

Figure 1: Generate logmatic time samples.

- Calculate the Heat Kernel Signature of all vertices (see Fig. 2).

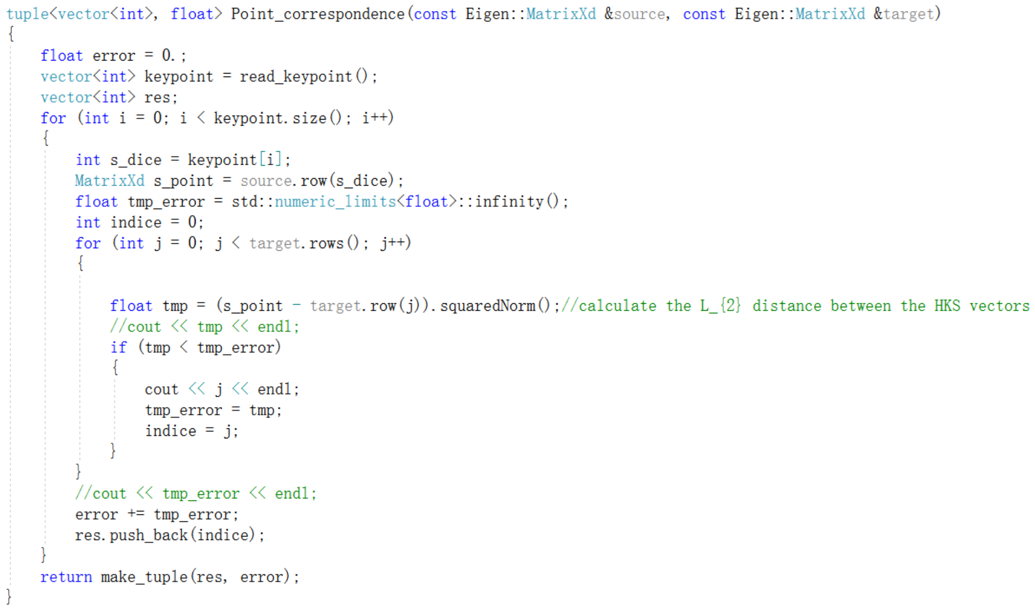


```
eg_vect = eg_vect.array().abs().square();//6890*300
//eg_val=-abs(eg_val.array());
eg_val = abs(eg_val(0, 0)) - abs(eg_val.array());//300*1

result = eg_vect * ((eg_val*seq).array().exp().matrix());

MatrixXd colsum = (diag*result).colwise().sum();//6890*6890\times 6890*100==>1*100
Eigen::SparseMatrix<double> scale(time_sample, time_sample);
for (int i = 0; i < time_sample; i++)
    scale.insert(i, i) = 1 / colsum(0, i);

result = result * scale;
saveData(root_path + "Results/" + name + "_hks.csv", result);
return result;
```

Figure 2: Calculate HKS of all vertices.

2 Keypoints Matching

Get the matched vertex in the target mesh by calculating the L_2 norm between the source HKS and the target HKS (**see Fig. 3**).

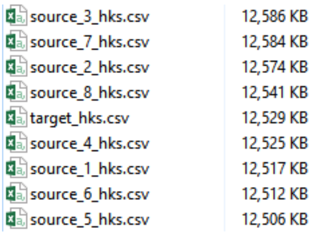


```
tuple<vector<int>, float> Point_correspondence(const Eigen::MatrixXd &source, const Eigen::MatrixXd &target)
{
    float error = 0.;
    vector<int> keypoint = read_keypoint();
    vector<int> res;
    for (int i = 0; i < keypoint.size(); i++)
    {
        int s_dice = keypoint[i];
        MatrixXd s_point = source.row(s_dice);
        float tmp_error = std::numeric_limits<float>::infinity();
        int indice = 0;
        for (int j = 0; j < target.rows(); j++)
        {

            float tmp = (s_point - target.row(j)).squaredNorm();//calculate the L_{2} distance between the HKS vectors
            //cout << tmp << endl;
            if (tmp < tmp_error)
            {
                cout << j << endl;
                tmp_error = tmp;
                indice = j;
            }
        }
        //cout << tmp_error << endl;
        error += tmp_error;
        res.push_back(indice);
    }
    return make_tuple(res, error);
}
```

Figure 3: Implement the point to point matching.

3 Implementation and Results.

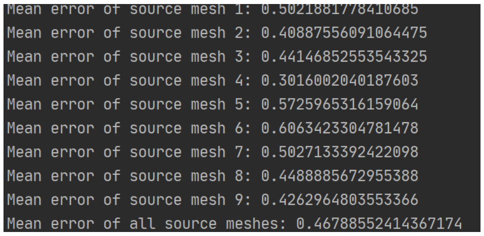
- You can implement the HKS calculation of all meshes and matching by running Report_total_error() function.
- Results of HKS calculation (using the scale version).



source_3_hks.csv	12,586 KB
source_7_hks.csv	12,584 KB
source_2_hks.csv	12,574 KB
source_8_hks.csv	12,541 KB
target_hks.csv	12,529 KB
source_4_hks.csv	12,525 KB
source_1_hks.csv	12,517 KB
source_6_hks.csv	12,512 KB
source_5_hks.csv	12,506 KB

Figure 4: Example Results.

- Evaluate the matching results (**see Fig. 5 Mean error:0.4678**).



```
Mean error of source mesh 1: 0.5021881778410685
Mean error of source mesh 2: 0.40887556091064475
Mean error of source mesh 3: 0.44146852553543325
Mean error of source mesh 4: 0.3016002040187603
Mean error of source mesh 5: 0.5725965316159064
Mean error of source mesh 6: 0.6063423304781478
Mean error of source mesh 7: 0.5027133392422098
Mean error of source mesh 8: 0.4488885672955388
Mean error of source mesh 9: 0.4262964803553366
Mean error of all source meshes: 0.46788552414367174
```

Figure 5: Example Results.