关于奇异值分解

1.对 A 进行奇异值分解,从 A^TA 分解出左右奇异向量:

$$Av_i = \sigma_i u_i \ ，\ \sigma_i(\text{奇异值})=\sqrt{\lambda_i} \ ，\ 1 \le i \le \mathrm{k} \ ，\ \mathrm{k}=\mathrm{Rank(A)}$$

该等式是把右奇异正交向量 v_i 映射成为左奇异正交向量 u_i,奇异值 sigmai 就是映射后的正交基的模的大小.奇异值的求取:

对任意mxn实矩阵A的奇异值分解A= USV^T ，有：

$$A^TA = VS^TSV^T$$

$$(A^TA)v_i = VS^TSV^Tv_i = \sigma_i^2 v_i$$

这不就是特征值分解吗。也就是说 v_i 是 A^TA 的特征向量，其对应的特征值是 σ_i^2 ，同理 u_i 是 AA^T 的特征向量，其对应的特征值也是 σ_i^2 。可以从这个角度出发求解特征值和特征向量。

注意: A^TA 特征值分解得到右奇异向量,AA^T 特征值分解得到左奇异向量

2. 比较特征值分解和奇异值分解

方阵的特征值分解:

$\mathbf{D}^\top\mathbf{D} = \sum_{i=1}^{4} \sigma_i^2 \mathbf{u}_i \mathbf{u}_j^\top$,这里的 u_i 是 D.transpose*D 的特征向量,也是对应 D 的右奇异向量

非方阵的奇异值分解:

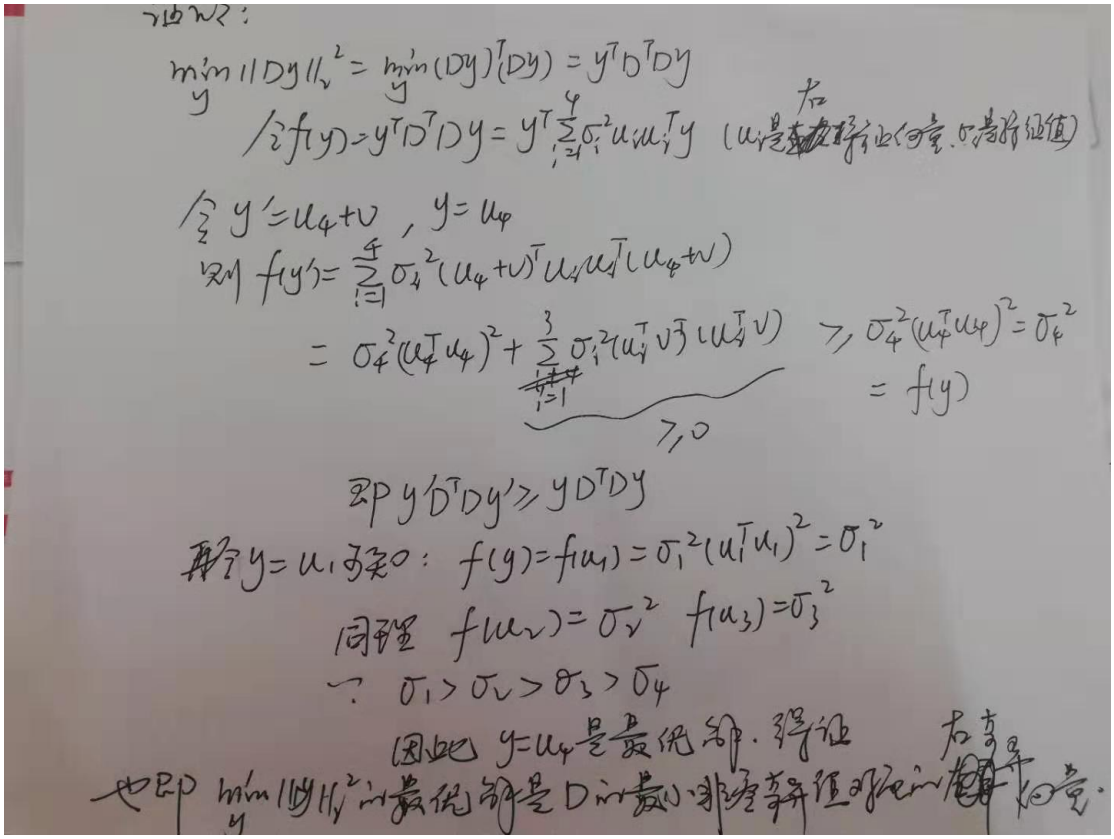
$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T$,这里的 u_i 对应 A 的左奇异向量,v_i 对应 A 的右奇异向量

3. 奇异值,秩,零空间的关系

A. 非零奇异值的个数就是矩阵的秩

B. 由于 r(A)+r(零空间)=n(A 的列数),因此矩阵零空间的维数就是 0 奇异值的个数

4. 作业,证明:



5. 代码块:

这里要注意，y 要大于 0，因此不能直接 llt 求解或者 svd 求解，因为 D 很容易满秩，从而解出来的结果是 0。根据理论，此时有两种解法：

1. 求 D.transpose*D 的特征向量，注意特征向量的排列是按照特征值的从小到大由左至右排列，因此选取第一列。

2. 求 D 的右奇异向量，奇异向量的排列是按照奇异值从大到小由左至右排列，因此选最后一列。

```
Eigen::MatrixXd D(2*(end_frame_id-start_frame_id),4);
Eigen::VectorXd b(Eigen::VectorXd::Zero(2*(end_frame_id-start_frame_id)));
for(int i=start_frame_id;i<poseNums;i++){
	Eigen::Matrix3d Rcw = camera_pose[i].Rwc.transpose();
	Eigen::Vector3d tcw = -Rcw * camera_pose[i].twc;
	Eigen::Matrix4d T=Eigen::Matrix4d::Zero();
	T.topLeftCorner(3,3) = Rcw;
	T.topRightCorner(3,1) = tcw;
	T(3,3) = 1;
```

```
      D.block((i-start_frame_id)*2,0,1,4) = camera_pose[i].uv[0]*T.row(2) -
T.row(0);
      D.block((i-start_frame_id)*2+1,0,1,4) = camera_pose[i].uv[1]*T.row(2) -
T.row(1);
   }
   std::cout<<D<<std::endl;
   Eigen::MatrixXd DTD = D.transpose()*D;
//    Eigen::Vector4d y=DTD.llt().solve(D.transpose() * b);// llt 分解要求 D 是正
定阵

   ///直接求解
//    Eigen::JacobiSVD<Eigen::MatrixXd> svd(D,Eigen::ComputeThinU |
Eigen::ComputeThinV);//构建最小二乘问题
//    Eigen::Vector4d y = svd.singularValues();
//    y=svd.solve(b);
   //std::cout<<y.transpose()<<std::endl;

   ///通过求 DTD 的特征向量
//    Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> eigenSolver(DTD);
//    std::cout<<std::endl;
//    std::cout<<eigenSolver.eigenvalues()<<std::endl;
//    Eigen::Matrix4d U = eigenSolver.eigenvectors();
//    std::cout<<std::endl;
//    std::cout<<U<<std::endl;
//    Eigen::Vector4d leastEigenVec = U.col(0);
//    P_est = leastEigenVec.head(3)/leastEigenVec[3];

   ///通过求 D 的奇异向量
   Eigen::JacobiSVD<Eigen::MatrixXd> svd(D,Eigen::ComputeThinU |
Eigen::ComputeThinV);
   Eigen::Matrix4d V = svd.matrixV();
   std::cout<<std::endl;
   std::cout<<V<<std::endl;
   Eigen::Vector4d leastEigenVec = V.col(3);
   P_est = leastEigenVec.head(3)/leastEigenVec[3];
```