**Home Assignment 1**

Daniel Smotritsky, Ilana Pervoi

1. **Vector norms, matrix norms, and inner products**
2. Since as it is the maximum sum of a column (in absolute values) of A, we want a normalized vector that multiplying the matrix by it will result this column.

is a vector that satisfies that:

Since as it is the maximum sum of a row (in absolute values) of A, we want a normalized vector that multiplying the matrix by it will result a vector with 18 as the largest entrance in it obtained from the sum of the row above (again, in absolute values).

is a vector that satisfies that:

1. Since , for convenience like in the previous section, we will restrict ourselves to , hence we need to find x such that ,

or equivalently:

Meaning x is the eigenvector that corresponds to the largest eigenvalue of (and since is always symmetric, its basis is orthonormal therefore its eigenvector normalized - ).

A = np.array([[1, 2, 3, 4], [2, 4, -4, 8], [-5, 4, 1, 5], [5, 0, -3, -7]])  
ATA = np.transpose(A) @ A

**# eigenvectors and eigenvalues of ATA**  
eigvals, eigvecs = scipy.linalg.eig(ATA)  
maxEigenVal = max(eigvals)

**# the index of the largest eigenvalue in eigenvals  
# the columns in eigenvecs corresponds to eigenvals**  
index = np.where(eigvals == maxEigenVal)  
x = eigvecs[:, index[0]]

print(x)

1. **The existence of solution for LS**
2. We need to prove that .

1. We need to prove that
2. We need to show that there always exists x such that i.e. the normal equation is always consistent.

meaning there is a vector x that satisfies .

1. **Least Squares**
2. We need to solve by least squares approximation, meaning we need to solve the normal equation .

Since is always symmetric matrix we can use Cholesky factorization on it and obtain .

Now by forward substitution we can found y and then by back substitution x.

AT = np.transpose(A)  
L = np.linalg.cholesky(AT @ A)  
  
**# A^TA=LL^T from the factorization  
# denote y = L^Tx  
# L in lower triangular - solve by forward substitution**  
y = scipy.linalg.solve\_triangular(L, AT @ b, lower=True)  
  
**# L^T in upper triangular - solve by back substitution**  
x = scipy.linalg.solve\_triangular(np.transpose(L), y, lower=False)  
print(x)

1. Now, we need to solve the problem by two other factorization methods:

* QR factorization of A:

Since A is real square matrix, it may be composed as where Q is orthogonal matrix and R is upper triangular matrix.

We get the next normal equation:

We can “divide from the right” in both sides since R is invertible (as A is invertible).

Finally, by back substitution we found x as in previous section.

**# QR factorization**  
Q, R = np.linalg.qr(A)  
QTb = np.transpose(Q) @ b  
  
**# Rx=QTb  
# R is upper triangular - solve by back substitution**  
x = scipy.linalg.solve\_triangular(R, QTb, lower=False)  
print(x)

* SVD factorization of A:

A (and any other matrix) can be composed as where U and V are unitary matrices and is diagonal with singular values of A.

We get the next normal equation:

Since V and Σ are square and invertible, we can “divide from the left”.

By solving the diagonal system, we get y.

Since V is square, invertible, and unitary .

Once again, we’ve got :

**# SVD factorization**  
U, S, VT = np.linalg.svd(A, full\_matrices=False)  
UTb = np.transpose(U) @ b

**# SVTx=UTb the normal equation  
# denote y=VTx  
# Sy=UTb is a diagonal system  
# y = (1/S)UTb trivial inversion of a diagonal matrix**  
y = np.diag(1 / S) @ UTb

**# x = VT**  
x = np.transpose(VT) @ y  
print(x)

1. The residual of the LS system with x we’ve found is

Not surprisingly we obtain Since we are looking for a solution x for the normal equation:

1. By taking a diagonal positive matrix that approximates the first equation in 808 times the accuracy, the solution of the weighted least squares normal equation results

and by computing the residual , the first equation is satisfied such that its first entry – r1, obtains:

as requested in the task.

W = np.eye(4, dtype=int)  
AT = np.transpose(A)  
ATWA = AT @ W @ A  
ATWb = AT @ W @ b

**# ATWAx=ATWb the normal equation**  
x = np.linalg.solve(ATWA, ATWb)  
r1 = (A @ x - b)[0]

while np.absolute(r1) >= pow(10, -3):  
 W[0, 0] += 1  
 **# re-calculate**  
 ATWA = AT @ W @ A  
 ATWb = AT @ W @ b  
 x = np.linalg.solve(ATWA, ATWb)  
 r1 = (A @ x - b)[0]  
   
print(W)  
print(r1)  
print(x)

1. **QR factorization**

**# QR factorization using GS**  
**def qr\_grams(a):**  
 n = len(a[0])

**# initialize**  
 R = np.zeros((n, n))  
 a0 = a[:, 0]  
 R[0, 0] = np.linalg.norm(a0)  
 q0 = a0 / R[0, 0]  
 Q = np.array([q0]).T

for i in range(1, n):  
 ai = a[:, i]  
 qi = ai  
 for j in range(0, i):  
 qj = Q[:, j]  
 R[j, i] = np.transpose(qj) @ **ai**  
 qi = qi - (R[j, i] \* qj)  
 R[i, i] = np.linalg.norm(qi)  
 qi = qi / R[i, i]  
 Q = np.column\_stack((Q, qi))

return Q, R

**# QR factorization using MGS**  
**def qr\_mod\_grams(a):**  
 n = len(a[0])

**# initialize**  
 R = np.zeros((n, n))  
 a0 = a[:, 0]  
 R[0, 0] = np.linalg.norm(a0)  
 q0 = a0 / R[0, 0]  
 Q = np.array([q0]).T

for i in range(1, n):  
 ai = a[:, i]  
 qi = ai  
 for j in range(0, i):  
 qj = Q[:, j]  
 **# difference from regular gs in dependence**

**on changed qi**  
 R[j, i] = np.transpose(qj) @ **qi**  
 qi = qi - (R[j, i] \* qj)  
 R[i, i] = np.linalg.norm(qi)  
 qi = qi / R[i, i]  
 Q = np.column\_stack((Q, qi))

return Q, R

1. For , we obtain both in regular Gram-Schmidt and Modified Gram-Schmidt the factorization:

For we obtain:

Regular GS:

Modified GS:

1. For ,

from regular GS we obtain

and from MGS :

for ,

from regular GS we obtain

and from MGS :

We can notice that when , the regular factorization resulted better results in contrary to the results when , in which the modified factorization produced better precision of the matrix Q to be orthogonal.

This phenomenon occurs because the regular GS algorithm can be numerically unstable, meaning it is sensitive to rounding error which leads to high numerical errors and eventually result the orthogonality of the computed vectors to be lost. The MGS on the other hand consistently produces vectors which are more orthogonal than those generated by regular GS since they consider the change of .

In the case of , the values in the matrix are more “stable” hence the difference between the algorithms results is very small (1.1290054305274943e-16), but when , in the regular GS algorithm if an error is made in computing  for example, so that is small (not 0), it will not be corrected for in any if the following computations leading to further losses of orthogonality due to cancellations in the computation that may occur.

1. **Regularized Least Squares and SVD**
2. We need to show that .

From the previous section we know that:

We will observe this equation:

=

=

1. In this question we have

We need to prove:

From (b) we obtain:

We will observe:

=

Diagnosis:

U is a unitary matrix, thus all the vectors in it are orthogonal to each other.

Meaning:

Therefore, the only expressions that are left are

1. We need to show that using regularized LS,
2. The normal equation is
3. is always PD for (without assuming A is full rank).

By setting the derivative to 0 we will obtain the normal equation (as taught in class):

Now, assuming :

Also:

Let v and be eigen-vector and eigen-value of .

Thus, v is eigen-vector of that corresponds to the eigen-value .

Let u be a vector (not zero).

If so, its eigen-values are not negative, so do .

We assumed that

1. We need to show that the solution to regularized LS is

Decomposing A with SVD gives us the following:

Hence, we get the next normal equation:

Notice that is a diagonal matrix with positive diagonal, meaning positive eigen-values, thus it is non-singular.

We will observe the matrix :

It is diagonal too, so by inserting its eigen-values in the form from section (b):

and finally, from section (c):

1. In the deblurring example, under the assumption that the noise is associated with a single vector with a small singular value the result of the regular LS as seen in (c) might be very large ( when ), meaning it is not a good approximation for our problem.

On the other hand, as seen in (e), the result of the regularized LS computed by is being regularized when choosing since it balances the expression to not be as large as seen in the regular LS when is small, therefore we get a better solution to our problem resulting a sharper picture.

1. **Camera Calibration**
2. The minimal number of correspondences required is **2**.

Explanation:

We need to find 4 parameters – .

From the given model:

we obtain 2 linear equations:

As we already know, to determine 4 parameters, we need at least 4 equations in our linear system.

Since having one set of correspondences provides two equations, 2 sets will provide 4 equations as required.

1. By representing the linear system from the previous section as matrices we obtain:

Assuming sets of correspondences:

A good solution to the problem obtained by solving the system in a LS sense where:

We can see that is invertible