# Reading in the data

The following code is used to read in the data

// read in data

    ifstream i\_A\_data("Ex1A.dat");

    // read out M and N

    int M, N;

    i\_A\_data >> M;

    i\_A\_data >> N;

    std::cout << "M: " << M << " N: " << N << std::endl;

    VecDoub i\_matrix\_1(40);

    VecDoub i\_matrix\_2(40);

    VecDoub i\_matrix\_3(40);

    VecDoub i\_matrix\_4(40);

    VecDoub i\_matrix\_5(40);

    VecDoub i\_matrix\_6(40);

    for (int i = 0; i < 40; i++)

    {

        i\_A\_data >> i\_matrix\_1[i];

        i\_A\_data >> i\_matrix\_2[i];

        i\_A\_data >> i\_matrix\_3[i];

        i\_A\_data >> i\_matrix\_4[i];

        i\_A\_data >> i\_matrix\_5[i];

        i\_A\_data >> i\_matrix\_6[i];

    }

    MatDoub A(40, 6);

    for (int i = 0; i < 40; i++)

    {

        A[i][0] = i\_matrix\_1[i];

        A[i][1] = i\_matrix\_2[i];

        A[i][2] = i\_matrix\_3[i];

        A[i][3] = i\_matrix\_4[i];

        A[i][4] = i\_matrix\_5[i];

        A[i][5] = i\_matrix\_6[i];

    }

    util::print(A, "A");

    // read data

    ifstream i\_b\_data("Ex1B.dat");

    // read out M and N

    i\_b\_data >> M;

    i\_b\_data >> N;

    std::cout << "M: " << M << " N: " << N << std::endl;

    VecDoub b(M);

    for (int i = 0; i < M; i++)

    {

        i\_b\_data >> b[i];

    }

    util::print(b, "b");

# Exercise 1

### i)

The code used to generate the result is the following:

    // make SVD

    SVD svd\_exi(A);

    VecDoub xSvd\_exi(6);

    svd\_exi.solve(b, xSvd\_exi, THRESHOLD);

    cout << endl;

    util::print(svd\_exi.w, "svd\_w\_exi");

    cout << endl;

The diagonal elements of W are:

A screenshot of a computer screen

Description automatically generated

ii)

The used code for this exercise, is the same as for the one above. Now I just print the X instead of w

util::print(xSvd\_exi, "svd\_X\_exi");

The solution/X is:

A screenshot of a computer

Description automatically generated

iii)

The errors were found by using the following formula.

A math equation with numbers and symbols

Description automatically generated

This was implemented with the use of two for loops counting through V and W, as seen in the code below.

VecDoub errors(SVD svd)

{

    VecDoub errors(svd.n);

    for (int j = 0; j < svd.n; j++)

    {

        double error = 0;

        for (int i = 0; i < svd.n; i++)

        {

            error += (svd.v[j][i] / svd.w[i]) \* (svd.v[j][i] / svd.w[i]);

        }

        errors[j] = sqrt(error);

    }

    return errors;

}

int main()

{

    VecDoub errors\_exi = svd\_exi.errors();

    util::print(errors\_exi, "errors");

The answers were the following:

A screenshot of a computer screen

Description automatically generated

iV)

The residuals were found with the following code:

    VecDoub residuals\_exi = A \* xSvd\_exi - b;

    util::print(residuals\_exi, "residuals");

where it should be noted that I have added the following function to utilities

VecDoub operator-(const VecDoub &a, const VecDoub &b)

{

    if (a.size() != b.size())

    {

        cerr << "in prod: the size of vector a is not equal to the size of vector b" << endl;

    }

    VecDoub res(a.size());

    for (int n = 0; n < a.size(); n++)

    {

        res[n] = a[n] - b[n];

    }

    return res;

}

This gives the following answer:

A screenshot of a computer screen

Description automatically generated

v)

The used code can be seen below:

VecDoub sigma\_i(double delta, VecDoub residuals)

{

    VecDoub sigma\_i(residuals.size());

    for (int i = 0; i < residuals.size(); i++)

    {

        sigma\_i[i] = abs(residuals[i]) < delta ? delta : abs(residuals[i]);

    }

    return sigma\_i;

}

    VecDoub sigma = sigma\_i(1, residuals\_exi);

    util::print(sigma, "sigma");

    // create new A

    MatDoub A\_sigma = A;

    for (int i = 0; i < sigma.size(); i++)

    {

        for (int j = 0; j < 6; j++)

        {

            A\_sigma[i][j] /= sigma[i];

        }

    }

    util::print(A\_sigma, "A\_sigma");

    // create new b

    VecDoub b\_sigma = b;

    for (int i = 0; i < sigma.size(); i++)

    {

        b\_sigma[i] /= sigma[i];

    }

    util::print(b\_sigma, "b\_sigma");

    std::cout << "b\_sigma[6]:" << b\_sigma[6] << endl;

    std::cout << "A\_sigma[0][0]:" << A\_sigma[0][0] << endl;

The two values can be seen here:



vi)

The used code can be seen here:

    // make SVD on new values

    SVD svd\_sigma\_exi(A\_sigma);

    VecDoub xSvd\_sigma\_exi(6);

    svd\_sigma\_exi.solve(b\_sigma, xSvd\_sigma\_exi, THRESHOLD);

    cout << endl;

    util::print(xSvd\_sigma\_exi, "x\_svd\_exi");

    cout << endl;

With the new matrices the solution x is:

A screenshot of a computer screen

Description automatically generated