Reading in the data

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:

```
svd_w_exi          Vector 6D:
          4752.37
          806.789
          58.1826
          16.9671
           3.5654
          3.02446
```

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:

```
svd_X_exi        Vector 6D:
        -24.3946
         72.8921
          1.7861
        -63.2459
        -76.9598
        -17.3859
```

iii)

The errors were found by using the following formula.

$$[\delta \mathbf{x}]_j \simeq \sqrt{\sum_{i=1}^{n} \left(\frac{V_{ji}}{w_i} \right)^2} \qquad j = 1, \ldots, n$$

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:

```
errors   Vector 6D:
        0.322198
         0.27585
       0.0960741
          0.0509
      0.00522238
      0.00210278
```

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:

```
Exercise iv

residuals        Vector 40D:
       -8.55671
       0.527233
      -0.673201
       -1.26463
      -0.831363
       0.464874
         7.6186
        1.13496
      -0.401232
      -0.325707
      -0.217701
       0.230878
      -0.218717
       0.656575
     -0.0388285
      -0.535747
       0.752515
       0.561871
       0.324391
      -0.743685
      -0.744122
         1.0811
      -0.605121
      -0.362054
       0.341041
       -0.89615
      -0.411017
     -0.0485407
        1.13667
        1.08557
      -0.627664
      -0.400131
        1.06093
       0.132202
        0.55506
       0.408264
        0.87311
      -0.680265
       0.400403
      -0.763662
```

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:

```
b_sigma[6]:284.403
A_sigma[0][0]:0.116867
```

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:

```
x_svd_exi       Vector 6D:
        -25.3757
         72.8928
         2.01114
         -63.252
        -76.9682
        -17.3859
```