Assignment 11

The program for SVD is taken from the link
http://www.public.iastate.edu/~dicook/JSS/paper/code/svd.c
for your understanding.

1. **You are supposed to run (after modifying, if needed) it using either C or Fortran and try to find the SVD of suitable real and complex matrices (with minimum dimension of 5 on a side, e.g., 10 x 5, 6 x 5 etc).**
2. **Try to verify your results using higher level software.**

```
/*
 * svdcomp - SVD decomposition routine.
 * Takes an mxn matrix a and decomposes it into udv, where u,v are
 * left and right orthogonal transformation matrices, and d is a
 * diagonal matrix of singular values.
 *
 * This routine is adapted from svdecomp.c in XLISP-STAT 2.1 which is
 * code from Numerical Recipes adapted by Luke Tierney and David Betz.
 *
 * Input to dsvd is as follows:
 *   a = mxn matrix to be decomposed, gets overwritten with u
 *   m = row dimension of a
 *   n = column dimension of a
 *   w = returns the vector of singular values of a
 *   v = returns the right orthogonal transformation matrix
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "defs_and_types.h"

static double PYTHAG(double a, double b)
{
    double at = fabs(a), bt = fabs(b), ct, result;

    if (at > bt)       { ct = bt / at; result = at * sqrt(1.0 + ct * ct); }
    else if (bt > 0.0) { ct = at / bt; result = bt * sqrt(1.0 + ct * ct); }
    else result = 0.0;
    return(result);
}


int dsvd(float **a, int m, int n, float *w, float **v)
{
    int flag, i, its, j, jj, k, l, nm;
    double c, f, h, s, x, y, z;
    double anorm = 0.0, g = 0.0, scale = 0.0;
    double *rv1;

    if (m < n)
    {
        fprintf(stderr, "#rows must be > #cols \n");
        return(0);
```

```
    }

    rv1 = (double *)malloc((unsigned int) n*sizeof(double));

/* Householder reduction to bidiagonal form */
    for (i = 0; i < n; i++)
    {
        /* left-hand reduction */
        l = i + 1;
        rv1[i] = scale * g;
        g = s = scale = 0.0;
        if (i < m)
        {
            for (k = i; k < m; k++)
                scale += fabs((double)a[k][i]);
            if (scale)
            {
                for (k = i; k < m; k++)
                {
                    a[k][i] = (float)((double)a[k][i]/scale);
                    s += ((double)a[k][i] * (double)a[k][i]);
                }
                f = (double)a[i][i];
                g = -SIGN(sqrt(s), f);
                h = f * g - s;
                a[i][i] = (float)(f - g);
                if (i != n - 1)
                {
                    for (j = l; j < n; j++)
                    {
                        for (s = 0.0, k = i; k < m; k++)
                            s += ((double)a[k][i] * (double)a[k][j]);
                        f = s / h;
                        for (k = i; k < m; k++)
                            a[k][j] += (float)(f * (double)a[k][i]);
                    }
                }
                for (k = i; k < m; k++)
                    a[k][i] = (float)((double)a[k][i]*scale);
            }
        }
        w[i] = (float)(scale * g);

        /* right-hand reduction */
        g = s = scale = 0.0;
        if (i < m && i != n - 1)
        {
            for (k = l; k < n; k++)
                scale += fabs((double)a[i][k]);
            if (scale)
            {
                for (k = l; k < n; k++)
                {
                    a[i][k] = (float)((double)a[i][k]/scale);
                    s += ((double)a[i][k] * (double)a[i][k]);
                }
                f = (double)a[i][l];
```

```
                g = -SIGN(sqrt(s), f);
                h = f * g - s;
                a[i][l] = (float)(f - g);
                for (k = l; k < n; k++)
                    rv1[k] = (double)a[i][k] / h;
                if (i != m - 1)
                {
                    for (j = l; j < m; j++)
                    {
                        for (s = 0.0, k = l; k < n; k++)
                            s += ((double)a[j][k] * (double)a[i][k]);
                        for (k = l; k < n; k++)
                            a[j][k] += (float)(s * rv1[k]);
                    }
                }
                for (k = l; k < n; k++)
                    a[i][k] = (float)((double)a[i][k]*scale);
            }
        }
        anorm = MAX(anorm, (fabs((double)w[i]) + fabs(rv1[i])));
    }

    /* accumulate the right-hand transformation */
    for (i = n - 1; i >= 0; i--)
    {
        if (i < n - 1)
        {
            if (g)
            {
                for (j = l; j < n; j++)
                    v[j][i] = (float)(((double)a[i][j] / (double)a[i][l]) /
g);
                    /* double division to avoid underflow */
                for (j = l; j < n; j++)
                {
                    for (s = 0.0, k = l; k < n; k++)
                        s += ((double)a[i][k] * (double)v[k][j]);
                    for (k = l; k < n; k++)
                        v[k][j] += (float)(s * (double)v[k][i]);
                }
            }
            for (j = l; j < n; j++)
                v[i][j] = v[j][i] = 0.0;
        }
        v[i][i] = 1.0;
        g = rv1[i];
        l = i;
    }

    /* accumulate the left-hand transformation */
    for (i = n - 1; i >= 0; i--)
    {
        l = i + 1;
        g = (double)w[i];
        if (i < n - 1)
            for (j = l; j < n; j++)
                a[i][j] = 0.0;
```

```
    if (g)
    {
        g = 1.0 / g;
        if (i != n - 1)
        {
            for (j = l; j < n; j++)
            {
                for (s = 0.0, k = l; k < m; k++)
                    s += ((double)a[k][i] * (double)a[k][j]);
                f = (s / (double)a[i][i]) * g;
                for (k = i; k < m; k++)
                    a[k][j] += (float)(f * (double)a[k][i]);
            }
        }
        for (j = i; j < m; j++)
            a[j][i] = (float)((double)a[j][i]*g);
    }
    else
    {
        for (j = i; j < m; j++)
            a[j][i] = 0.0;
    }
    ++a[i][i];
}

/* diagonalize the bidiagonal form */
for (k = n - 1; k >= 0; k--)
{                               /* loop over singular values */
    for (its = 0; its < 30; its++)
    {                           /* loop over allowed iterations */
        flag = 1;
        for (l = k; l >= 0; l--)
        {                       /* test for splitting */
            nm = l - 1;
            if (fabs(rv1[l]) + anorm == anorm)
            {
                flag = 0;
                break;
            }
            if (fabs((double)w[nm]) + anorm == anorm)
                break;
        }
        if (flag)
        {
            c = 0.0;
            s = 1.0;
            for (i = l; i <= k; i++)
            {
                f = s * rv1[i];
                if (fabs(f) + anorm != anorm)
                {
                    g = (double)w[i];
                    h = PYTHAG(f, g);
                    w[i] = (float)h;
                    h = 1.0 / h;
                    c = g * h;
                    s = (- f * h);
```

```
                    for (j = 0; j < m; j++)
                    {
                        y = (double)a[j][nm];
                        z = (double)a[j][i];
                        a[j][nm] = (float)(y * c + z * s);
                        a[j][i] = (float)(z * c - y * s);
                    }
                }
            }
        }
        z = (double)w[k];
        if (l == k)
        {                   /* convergence */
            if (z < 0.0)
            {               /* make singular value nonnegative */
                w[k] = (float)(-z);
                for (j = 0; j < n; j++)
                    v[j][k] = (-v[j][k]);
            }
            break;
        }
        if (its >= 30) {
            free((void*) rv1);
            fprintf(stderr, "No convergence after 30,000! iterations
\n");
            return(0);
        }

        /* shift from bottom 2 x 2 minor */
        x = (double)w[l];
        nm = k - 1;
        y = (double)w[nm];
        g = rv1[nm];
        h = rv1[k];
        f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0 * h * y);
        g = PYTHAG(f, 1.0);
        f = ((x - z) * (x + z) + h * ((y / (f + SIGN(g, f))) - h)) / x;

        /* next QR transformation */
        c = s = 1.0;
        for (j = l; j <= nm; j++)
        {
            i = j + 1;
            g = rv1[i];
            y = (double)w[i];
            h = s * g;
            g = c * g;
            z = PYTHAG(f, h);
            rv1[j] = z;
            c = f / z;
            s = h / z;
            f = x * c + g * s;
            g = g * c - x * s;
            h = y * s;
            y = y * c;
            for (jj = 0; jj < n; jj++)
            {
```

```
                    x = (double)v[jj][j];
                    z = (double)v[jj][i];
                    v[jj][j] = (float)(x * c + z * s);
                    v[jj][i] = (float)(z * c - x * s);
                }
                z = PYTHAG(f, h);
                w[j] = (float)z;
                if (z)
                {
                    z = 1.0 / z;
                    c = f * z;
                    s = h * z;
                }
                f = (c * g) + (s * y);
                x = (c * y) - (s * g);
                for (jj = 0; jj < m; jj++)
                {
                    y = (double)a[jj][j];
                    z = (double)a[jj][i];
                    a[jj][j] = (float)(y * c + z * s);
                    a[jj][i] = (float)(z * c - y * s);
                }
            }
            rv1[l] = 0.0;
            rv1[k] = f;
            w[k] = (float)x;
        }
    }
    free((void*) rv1);
    return(1);
}
```