NUMERICAL RECIPES

Webnote No. 2, Rev. 1

SVD Implementation

We here list the implementation that constructs the singular value decomposition of any matrix. See §11.3–§11.4, and also [1,2], for discussion relating to the underlying method. Note that all the hard work is done by `decompose`; `reorder` simply orders the columns into canonical order (decreasing w_j's, and with sign flips to get the maximum number of positive elements. The function `pythag` does just what you might guess from its name, coded so as avoid overflow or underflow.

svd.h

```
void SVD::decompose() {
```

Given the matrix $\mathbf{A}$ stored in `u[0..m-1][0..n-1]`, this routine computes its singular value decomposition, $\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T$ and stores the results in the matrices `u` and `v`, and the vector `w`.

```
    bool flag;
    Int i,its,j,jj,k,l,nm;
    Doub anorm,c,f,g,h,s,scale,x,y,z;
    VecDoub rv1(n);
    g = scale = anorm = 0.0;                Householder reduction to bidiagonal form.
    for (i=0;i<n;i++) {
        l=i+2;
        rv1[i]=scale*g;
        g=s=scale=0.0;
        if (i < m) {
            for (k=i;k<m;k++) scale += abs(u[k][i]);
            if (scale != 0.0) {
                for (k=i;k<m;k++) {
                    u[k][i] /= scale;
                    s += u[k][i]*u[k][i];
                }
                f=u[i][i];
                g = -SIGN(sqrt(s),f);
                h=f*g-s;
                u[i][i]=f-g;
                for (j=l-1;j<n;j++) {
                    for (s=0.0,k=i;k<m;k++) s += u[k][i]*u[k][j];
                    f=s/h;
                    for (k=i;k<m;k++) u[k][j] += f*u[k][i];
                }
                for (k=i;k<m;k++) u[k][i] *= scale;
            }
        }
        w[i]=scale *g;
        g=s=scale=0.0;
        if (i+1 <= m && i+1 != n) {
            for (k=l-1;k<n;k++) scale += abs(u[i][k]);
            if (scale != 0.0) {
                for (k=l-1;k<n;k++) {
                    u[i][k] /= scale;
                    s += u[i][k]*u[i][k];
```

```
            }
            f=u[i][l-1];
            g = -SIGN(sqrt(s),f);
            h=f*g-s;
            u[i][l-1]=f-g;
            for (k=l-1;k<n;k++) rv1[k]=u[i][k]/h;
            for (j=l-1;j<m;j++) {
                for (s=0.0,k=l-1;k<n;k++) s += u[j][k]*u[i][k];
                for (k=l-1;k<n;k++) u[j][k] += s*rv1[k];
            }
            for (k=l-1;k<n;k++) u[i][k] *= scale;
        }
    }
    anorm=MAX(anorm,(abs(w[i])+abs(rv1[i])));
}
for (i=n-1;i>=0;i--) {                Accumulation of right-hand transformations.
    if (i < n-1) {
        if (g != 0.0) {
            for (j=l;j<n;j++)         Double division to avoid possible underflow.
                v[j][i]=(u[i][j]/u[i][l])/g;
            for (j=l;j<n;j++) {
                for (s=0.0,k=l;k<n;k++) s += u[i][k]*v[k][j];
                for (k=l;k<n;k++) v[k][j] += s*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;
}
for (i=MIN(m,n)-1;i>=0;i--) {         Accumulation of left-hand transformations.
    l=i+1;
    g=w[i];
    for (j=l;j<n;j++) u[i][j]=0.0;
    if (g != 0.0) {
        g=1.0/g;
        for (j=l;j<n;j++) {
            for (s=0.0,k=l;k<m;k++) s += u[k][i]*u[k][j];
            f=(s/u[i][i])*g;
            for (k=i;k<m;k++) u[k][j] += f*u[k][i];
        }
        for (j=i;j<m;j++) u[j][i] *= g;
    } else for (j=i;j<m;j++) u[j][i]=0.0;
    ++u[i][i];
}
for (k=n-1;k>=0;k--) {                Diagonalization of the bidiagonal form: Loop over
    for (its=0;its<30;its++) {            singular values, and over allowed iterations.
        flag=true;
        for (l=k;l>=0;l--) {          Test for splitting.
            nm=l-1;
            if (l == 0 || abs(rv1[l]) <= eps*anorm) {
                flag=false;
                break;
            }
            if (abs(w[nm]) <= eps*anorm) break;
        }
        if (flag) {
            c=0.0;                    Cancellation of rv1[l], if l > 0.
            s=1.0;
            for (i=l;i<k+1;i++) {
                f=s*rv1[i];
                rv1[i]=c*rv1[i];
                if (abs(f) <= eps*anorm) break;
```

```
                g=w[i];
                h=pythag(f,g);
                w[i]=h;
                h=1.0/h;
                c=g*h;
                s = -f*h;
                for (j=0;j<m;j++) {
                    y=u[j][nm];
                    z=u[j][i];
                    u[j][nm]=y*c+z*s;
                    u[j][i]=z*c-y*s;
                }
            }
        }
        z=w[k];
        if (l == k) {                   Convergence.
            if (z < 0.0) {              Singular value is made nonnegative.
                w[k] = -z;
                for (j=0;j<n;j++) v[j][k] = -v[j][k];
            }
            break;
        }
        if (its == 29) throw("no convergence in 30 svdcmp iterations");
        x=w[l];                         Shift from bottom 2-by-2 minor.
        nm=k-1;
        y=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;
        c=s=1.0;                        Next QR transformation:
        for (j=l;j<=nm;j++) {
            i=j+1;
            g=rv1[i];
            y=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 *= c;
            for (jj=0;jj<n;jj++) {
                x=v[jj][j];
                z=v[jj][i];
                v[jj][j]=x*c+z*s;
                v[jj][i]=z*c-x*s;
            }
            z=pythag(f,h);
            w[j]=z;                     Rotation can be arbitrary if z = 0.
            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=u[jj][j];
                z=u[jj][i];
```

```
                    u[jj][j]=y*c+z*s;
                    u[jj][i]=z*c-y*s;
                }
            }
            rv1[l]=0.0;
            rv1[k]=f;
            w[k]=x;
        }
    }
}

void SVD::reorder() {
```
Given the output of decompose, this routine sorts the singular values, and corresponding columns of u and v, by decreasing magnitude. Also, signs of corresponding columns are flipped so as to maximize the number of positive elements.
```
    Int i,j,k,s,inc=1;
    Doub sw;
    VecDoub su(m), sv(n);
    do { inc *= 3; inc++; } while (inc <= n);          Sort. The method is Shell's sort.
    do {                                                (The work is negligible as com-
        inc /= 3;                                       pared to that already done in
        for (i=inc;i<n;i++) {                           decompose.)
            sw = w[i];
            for (k=0;k<m;k++) su[k] = u[k][i];
            for (k=0;k<n;k++) sv[k] = v[k][i];
            j = i;
            while (w[j-inc] < sw) {
                w[j] = w[j-inc];
                for (k=0;k<m;k++) u[k][j] = u[k][j-inc];
                for (k=0;k<n;k++) v[k][j] = v[k][j-inc];
                j -= inc;
                if (j < inc) break;
            }
            w[j] = sw;
            for (k=0;k<m;k++) u[k][j] = su[k];
            for (k=0;k<n;k++) v[k][j] = sv[k];

        }
    } while (inc > 1);
    for (k=0;k<n;k++) {                                 Flip signs.
        s=0;
        for (i=0;i<m;i++) if (u[i][k] < 0.) s++;
        for (j=0;j<n;j++) if (v[j][k] < 0.) s++;
        if (s > (m+n)/2) {
            for (i=0;i<m;i++) u[i][k] = -u[i][k];
            for (j=0;j<n;j++) v[j][k] = -v[j][k];
        }
    }
}

Doub SVD::pythag(const Doub a, const Doub b) {
```
Computes $(a^2 + b^2)^{1/2}$ without destructive underflow or overflow.
```
    Doub absa=abs(a), absb=abs(b);
    return (absa > absb ? absa*sqrt(1.0+SQR(absb/absa)) :
        (absb == 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb))));
}
```

CITED REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 2002, *Introduction to Numerical Analysis*, 3rd ed. (New York: Springer), §6.7.[1]

Golub, G.H., and Van Loan, C.F. 1996, *Matrix Computations*, 3rd ed. (Baltimore: Johns Hopkins University Press), Chapter 12 (SVD).[2]