PROGRAM NO: 4

Input a matrix A and generate two orthogonal matrices U,V and a diagonal matrix D such that A=UDV^T

AIM

Write a program to Input a matrix A and generate two orthogonal matrices U,V and a diagonal matrix D such that A=UDV^T

PROGRAM

```
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <time.h>
# include <string.h>
# include "linpack_d.h"
# include "blas1_d.h"

int main ( int argc, char *argv[] );
int get_seed ( void );
double *pseudo_inverse ( int m, int n, double u[], double s[],
  double v[] );
void pseudo_linear_solve_test ( int m, int n, double a[],
  double a_pseudo[], int *seed );
void pseudo_product_test ( int m, int n, double a[], double a_pseudo[] );
int r8_nint ( double x );
double r8mat_dif_fro ( int m, int n, double a[], double b[] );
double r8mat_norm_fro ( int m, int n, double a[] );
void r8mat_print ( int m, int n, double a[], char *title );
void r8mat_print_some ( int m, int n, double a[], int ilo, int jlo, int ihi,
  int jhi, char *title );
void r8mat_svd_linpack ( int m, int n, double a[], double u[], double s[],
  double v[] );
double *r8mat_uniform_01_new ( int m, int n, int *seed );
double r8vec_norm_l2 ( int n, double a[] );
double *r8vec_uniform_01_new ( int n, int *seed );
void rank_one_print_test ( int m, int n, double a[], double u[],
  double s[], double v[] );
void rank_one_test ( int m, int n, double a[], double u[], double s[],
  double v[] );
int s_len_trim ( char *s );
void svd_product_test ( int m, int n, double a[], double u[],
  double s[], double v[] );
void timestamp ( );

int main ( int argc, char *argv[] )

{
```

```
double *a;
double *a_pseudo;
int i;
int j;
int m;
int n;
double *s;
int seed;
char string[80];
double *u;
double *v;

timestamp ( );

printf ( "\n" );
printf ( "SVD_DEMO:\n" );
printf ( "  C version\n" );
printf ( "\n" );
printf ( "  Compiled on %s at %s.\n", __DATE__, __TIME__ );
printf ( "\n" );
printf ( "  Demonstrate the singular value decomposition (SVD)\n" );
printf ( "\n" );
printf ( "  A real MxN matrix A can be factored as:\n" );
printf ( "\n" );
printf ( "    A = U * S * V'\n" );
printf ( "\n" );
printf ( "  where\n" );
printf ( "\n" );
printf ( "    U = MxM orthogonal,\n" );
printf ( "    S = MxN zero except for diagonal,\n" );
printf ( "    V = NxN orthogonal.\n" );
printf ( "\n" );
printf ( "  The diagonal of S contains only nonnegative numbers\n" );
printf ( "  and these are arranged in descending order.\n" );

if ( argc < 2 )
{
  printf ( "\n" );
  printf ( "SVD_DEMO:\n" );
  printf ( "  Please enter the value of M:\n" );
  printf ( "  (Number of rows in matrix A).\n" );
  printf ( "  (We prefer M <= 10!).\n" );
  scanf ( "d", &m );
}
else
{
  strcpy ( string, argv[1] );
  m = atoi ( string );
}
printf ( "\n" );
printf ( "  Matrix row order    M = %d\n", m );

if ( argc < 3 )
{
  printf ( "\n" );
  printf ( "SVD_DEMO:\n" );
  printf ( "  Please enter the value of N:\n" );
```

```
  printf ( "  (Number of columns in matrix A).\n" );
  printf ( "  (We prefer N <= 10!).\n" );
  scanf ( "%d", &n );
}
else
{
  strcpy ( string, argv[2] );
  n = atoi ( string );
}
printf ( "  Matrix column order N = %d\n", n );

if ( argc < 4 )
{
  seed = get_seed ( );
  printf ( "  Random number SEED    = %d\n", seed );
  printf ( "  (Chosen by the program.)\n" );
}
else
{
  strcpy ( string, argv[3] );
  seed = atoi ( string );
  printf ( "  Random number SEED    = %d\n", seed );
  printf ( "  (Chosen by the user.)\n" );
}

u = ( double * ) malloc ( m * m * sizeof ( double ) );
s = ( double * ) malloc ( m * n * sizeof ( double ) );
v = ( double * ) malloc ( n * n * sizeof ( double ) );

  printf ( "\n" );
printf ( "  We choose a \"random\" matrix A, with integral\n" );
printf ( "  values between 0 and 10.\n" );

a = r8mat_uniform_01_new ( m, n, &seed );

for ( j = 0; j < n; j++ )
{
  for ( i = 0; i < m; i++ )
  {
    a[i+m*j] = r8_nint ( 10.0 * a[i+m*j] );
  }
}
r8mat_print ( m, n, a, "  The matrix A:\n" );

r8mat_print ( m, m, u, "  The orthogonal factor U:" );

r8mat_print ( m, n, s, "  The diagonal factor S:" );

r8mat_print ( n, n, v, "  The orthogonal factor V:" );

svd_product_test ( m, n, a, u, s, v );

rank_one_test ( m, n, a, u, s, v );

rank_one_print_test ( m, n, a, u, s, v );

a_pseudo = pseudo_inverse ( m, n, u, s, v );
```

```
  r8mat_print ( n, m, a_pseudo, "  The pseudoinverse of A:" );

  pseudo_product_test ( m, n, a, a_pseudo );

  pseudo_linear_solve_test ( m, n, a, a_pseudo, &seed );

  free ( a );
  free ( a_pseudo );
  free ( s );
  free ( u );
  free ( v );

  printf ( "\n" );
  printf ( "SVD_DEMO:\n" );
  printf ( "  Normal end of execution.\n" );

  printf ( "\n" );
  timestamp ( );

  return 0;
}

int get_seed ( void )
{
  time_t clock;
  int i;
  int i4_huge = 2147483647;
  int ihour;
  int imin;
  int isec;
  int seed;
  struct tm *lt;
  time_t tloc;

  clock = time ( &tloc );
  lt = localtime ( &clock );

  ihour = lt->tm_hour;

  if ( 12 < ihour )
  {
    ihour = ihour - 12;
  }

  ihour = ihour - 1;

  imin = lt->tm_min;

  isec = lt->tm_sec;

  seed = isec + 60 * ( imin + 60 * ihour );

  seed = seed + 1;

  seed = ( int )
    ( ( ( double ) seed )
```

```
    * ( ( double ) i4_huge ) / ( 60.0 * 60.0 * 12.0 ) );

  if ( seed == 0 )
  {
    seed = 1;
  }

  return seed;
}

double *pseudo_inverse ( int m, int n, double u[], double s[],
  double v[] )
{
  double *a_pseudo;
  int i;
  int j;
  int k;
  double *sp;
  double *sput;

  sp = ( double * ) malloc ( n * m * sizeof ( double ) );
  for ( j = 0; j < m; j++ )
  {
    for ( i = 0; i < n; i++ )
    {
      if ( i == j && s[i+i*m] != 0.0 )
      {
        sp[i+j*n] = 1.0 / s[i+i*m];
      }
      else
      {
        sp[i+j*n] = 0.0;
      }
    }
  }

  sput = ( double * ) malloc ( n * m * sizeof ( double ) );
  for ( i = 0; i < n; i++ )
  {
    for ( j = 0; j < m; j++ )
    {
      sput[i+j*n] = 0.0;
      for ( k = 0; k < m; k++ )
      {
        sput[i+j*n] = sput[i+j*n] + sp[i+k*n] * u[j+k*m];
      }
    }
  }

  a_pseudo = ( double * ) malloc ( n * m * sizeof ( double ) );
  for ( i = 0; i < n; i++ )
  {
    for ( j = 0; j < m; j++ )
    {
      a_pseudo[i+j*n] = 0.0;
      for ( k = 0; k < n; k++ )
      {
```

```
        a_pseudo[i+j*n] = a_pseudo[i+j*n] + v[i+k*n] * sput[k+j*n];
      }
    }
  }

  free ( sp );

  return a_pseudo;
}


void pseudo_linear_solve_test ( int m, int n, double a[],
  double a_pseudo[], int *seed )

{
  double *bm;
  double *bn;
  int i;
  int j;
  double *rm;
  double *rn;
  double *xm1;
  double *xm2;
  double *xn1;
  double *xn2;

  printf ( "\n" );
  printf ( "PSEUDO_LINEAR_SOLVE_TEST\n" );

  xn1 = r8vec_uniform_01_new ( n, seed );
  for ( i = 0; i < n; i++ )
  {
    xn1[i] = r8_nint ( 10.0 * xn1[i] );
  }

  bm = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
    bm[i] = 0.0;
    for ( j = 0; j < n; j++ )
    {
      bm[i] = bm[i] + a[i+j*m] * xn1[j];
    }
  }

  xn2 = ( double * ) malloc ( n * sizeof ( double ) );
  for ( i = 0; i < n; i++ )
  {
    xn2[i] = 0.0;
    for ( j = 0; j < m; j++ )
    {
      xn2[i] = xn2[i] + a_pseudo[i+j*n] * bm[j];
    }
  }

  rm = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
```

```
{
  rm[i] = bm[i];
  for ( j = 0; j < n; j++ )
  {
    rm[i] = rm[i] - a[i+j*m] * xn2[j];
  }
}

printf ( "\n" );
printf ( "  Given:\n" );
printf ( "    b = A * x1\n" );
printf ( "  so that b is in the range of A, solve\n" );
printf ( "    A * x = b\n" );
printf ( "  using the pseudoinverse:\n" );
printf ( "    x2 = A+ * b.\n" );
printf ( "\n" );
printf ( "  Norm of x1 = %g\n", r8vec_norm_l2 ( n, xn1 ) );
printf ( "  Norm of x2 = %g\n", r8vec_norm_l2 ( n, xn2 ) );
printf ( "  Norm of residual = %g\n", r8vec_norm_l2 ( m, rm ) );

free ( bm );
free ( rm );
free ( xn1 );
free ( xn2 );

if ( n < m )
{
  printf ( "\n" );
  printf ( "  For N < M, most systems A*x=b will not be\n" );
  printf ( "  exactly and uniquely solvable, except in the\n" );
  printf ( "  least squares sense.\n" );
  printf ( "\n" );
  printf ( "  Here is an example:\n" );

  bm = r8vec_uniform_01_new ( m, seed );

  xn2 = ( double * ) malloc ( n * sizeof ( double ) );
  for ( i = 0; i < n; i++ )
  {
    xn2[i] = 0.0;
    for ( j = 0; j < m; j++ )
    {
      xn2[i] = xn2[i] + a_pseudo[i+j*n] * bm[j];
    }
  }

  rm = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
    rm[i] = bm[i];
    for ( j = 0; j < n; j++ )
    {
      rm[i] = rm[i] - a[i+j*m] * xn2[j];
    }
  }

  printf ( "\n" );
```

```
  printf ( "  Given b is NOT in the range of A, solve\n" );
  printf ( "    A * x = b\n" );
  printf ( "  using the pseudoinverse:\n" );
  printf ( "    x2 = A+ * b.\n" );
  printf ( "\n" );
  printf ( "  Norm of x2 = %g\n", r8vec_norm_l2 ( n, xn2 ) );
  printf ( "  Norm of residual = %g\n", r8vec_norm_l2 ( m, rm ) );

  free ( bm );
  free ( rm );
  free ( xn2 );
}

xm1 = r8vec_uniform_01_new ( m, seed );
for ( i = 0; i < m; i++ )
{
  xm1[i] = r8_nint ( 10.0 * xm1[i] );
}

bn = ( double * ) malloc ( n * sizeof ( double ) );
for ( i = 0; i < n; i++ )
{
  bn[i] = 0.0;
  for ( j = 0; j < m; j++ )
  {
    bn[i] = bn[i] + a[j+i*m] * xm1[j];
  }
}

xm2 = ( double * ) malloc ( m * sizeof ( double ) );
for ( i = 0; i < m; i++ )
{
  xm2[i] = 0.0;
  for ( j = 0; j < n; j++ )
  {
    xm2[i] = xm2[i] + a_pseudo[j+i*n] * bn[j];
  }
}

rn = ( double * ) malloc ( n * sizeof ( double ) );
for ( i = 0; i < n; i++ )
{
  rn[i] = bn[i];
  for ( j = 0; j < m; j++ )
  {
    rn[i] = rn[i] - a[j+i*m] * xm2[j];
  }
}
printf ( "\n" );
printf ( "  Given:\n" );
printf ( "    b = A' * x1\n" );
printf ( "  so that b is in the range of A', solve\n" );
printf ( "    A' * x = b\n" );
printf ( "  using the pseudoinverse:\n" );
printf ( "    x2 = A+' * b.\n" );
printf ( "\n" );
printf ( "  Norm of x1 = %g\n", r8vec_norm_l2 ( m, xm1 ) );
```

```
  printf ( "  Norm of x2 = %g\n", r8vec_norm_l2 ( m, xm2 ) );
  printf ( "  Norm of residual = %g\n", r8vec_norm_l2 ( n, rn ) );

  free ( bn );
  free ( rn );
  free ( xm1 );
  free ( xm2 );

  if ( m < n )
  {
    printf ( "\n" );
    printf ( "  For M < N, most systems A'*x=b will not be\n" );
    printf ( "  exactly and uniquely solvable, except in the\n" );
    printf ( "  least squares sense.\n" );
    printf ( "\n" );
    printf ( "  Here is an example:\n" );

    bn = r8vec_uniform_01_new ( n, seed );

    xm2 = ( double * ) malloc ( m * sizeof ( double ) );
    for ( i = 0; i < m; i++ )
    {
      xm2[i] = 0.0;
      for ( j = 0; j < n; j++ )
      {
        xm2[i] = xm2[i] + a_pseudo[j+i*n] * bn[j];
      }
    }

    rn = ( double * ) malloc ( n * sizeof ( double ) );
    for ( i = 0; i < n; i++ )
    {
      rn[i] = bn[i];
      for ( j = 0; j < m; j++ )
      {
        rn[i] = rn[i] - a[j+i*m] * xm2[j];
      }
    }

    printf ( "\n" );
    printf ( "  Given b is NOT in the range of A', solve\n" );
    printf ( "    A' * x = b\n" );
    printf ( "  using the pseudoinverse:\n" );
    printf ( "    x2 = A+ * b.\n" );
    printf ( "\n" );
    printf ( "  Norm of x2 = %g\n", r8vec_norm_l2 ( m, xm2 ) );
    printf ( "  Norm of residual = %g\n", r8vec_norm_l2 ( n, rn ) );

    free ( bn );
    free ( rn );
    free ( xm2 );
  }

  return;
}

void pseudo_product_test ( int m, int n, double a[], double a_pseudo[] )
```

```
{
  double *bmm;
  double *bmn;
  double *bnm;
  double *bnn;
  double dif1;
  double dif2;
  double dif3;
  double dif4;
  int i;
  int j;
  int k;

  printf ( "\n" );
  printf ( "PSEUDO_PRODUCT_TEST\n" );
  printf ( "\n" );
  printf ( "  The following relations MUST hold:\n" );
  printf ( "\n" );
  printf ( "   A  * A+ * A  = A\n" );
  printf ( "   A+ * A  * A+ = A+\n" );
  printf ( " ( A  * A+ ) is MxM symmetric;\n" );
  printf ( " ( A+ * A  ) is NxN symmetric\n" );

  bnn = ( double * ) malloc ( n * n * sizeof ( double ) );
  for ( i = 0; i < n; i++ )
  {
    for ( j = 0; j < n; j++ )
    {
      bnn[i+j*n] = 0.0;
      for ( k = 0; k < m; k++ )
      {
        bnn[i+j*n] = bnn[i+j*n] + a_pseudo[i+k*n] * a[k+j*m];
      }
    }
  }
  bmn = ( double * ) malloc ( m * n * sizeof ( double ) );

  for ( i = 0; i < m; i++ )
  {
    for ( j = 0; j < n; j++ )
    {
      bmn[i+j*m] = 0.0;
      for ( k = 0; k < n; k++ )
      {
        bmn[i+j*m] = bmn[i+j*m] + a[i+k*m] * bnn[k+j*n];
      }
    }
  }
  dif1 = r8mat_dif_fro ( m, n, a, bmn );

  free ( bmn );
  free ( bnn );

  bmm = ( double * ) malloc ( m * m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
```

```
  for ( j = 0; j < m; j++ )
  {
    bmm[i+j*m] = 0.0;
    for ( k = 0; k < n; k++ )
    {
      bmm[i+j*m] = bmm[i+j*m] + a[i+k*m] * a_pseudo[k+j*n];
    }
  }
}

bnm = ( double * ) malloc ( n * m * sizeof ( double ) );

for ( i = 0; i < n; i++ )
{
  for ( j = 0; j < m; j++ )
  {
    bnm[i+j*n] = 0.0;
    for ( k = 0; k < m; k++ )
    {
      bnm[i+j*n] = bnm[i+j*n] + a_pseudo[i+k*n] * bmm[k+j*m];
    }
  }
}

dif2 = r8mat_dif_fro ( n, m, a_pseudo, bnm );

free ( bnm );
free ( bmm );

bmm = ( double * ) malloc ( m * m * sizeof ( double ) );
for ( i = 0; i < m; i++ )
{
  for ( j = 0; j < m; j++ )
  {
    bmm[i+j*m] = 0.0;
    for ( k = 0; k < n; k++ )
    {
      bmm[i+j*m] = bmm[i+j*m] + a[i+k*m] * a_pseudo[k+j*n];
    }
  }
}
dif3 = 0.0;
for ( j = 0; j < m; j++ )
{
  for ( i = 0; i < m; i++ )
  {
    dif3 = dif3 + pow ( bmm[i+j*m] - bmm[j+i*m], 2 );
  }
}
dif3 = sqrt ( dif3 );

free ( bmm );

bnn = ( double * ) malloc ( n * n * sizeof ( double ) );
for ( i = 0; i < n; i++ )
{
  for ( j = 0; j < n; j++ )
```

```
  {
    bnn[i+j*n] = 0.0;
    for ( k = 0; k < m; k++ )
    {
      bnn[i+j*n] = bnn[i+j*n] + a_pseudo[i+k*n] * a[k+j*m];
    }
  }
}
dif4 = 0.0;
for ( j = 0; j < n; j++ )
{
  for ( i = 0; i < n; i++ )
  {
    dif4 = dif4 + pow ( bnn[i+j*n] - bnn[j+i*n], 2 );
  }
}
dif4 = sqrt ( dif4 );

free ( bnn );

printf ( "\n" );
printf ( "  Here are the Frobenius norms of the errors\n" );
printf ( "  in these relationships:\n" );
printf ( "\n" );
printf ( "   A  * A+ * A  = A           %g\n", dif1 );
printf ( "   A+ * A  * A+ = A+          %g\n", dif2 );
printf ( " ( A  * A+ ) is MxM symmetric; %g\n", dif3 );
printf ( " ( A+ * A  ) is NxN symmetric; %g\n", dif4 );

printf ( "\n" );
printf ( "  In some cases, the matrix A * A+\n" );
printf ( "  may be interesting (if M <= N, then\n" );
printf ( "  it MIGHT look like the identity.)\n" );
printf ( "\n" );
bmm = ( double * ) malloc ( m * m * sizeof ( double ) );
for ( i = 0; i < m; i++ )
{
  for ( j = 0; j < m; j++ )
  {
    bmm[i+j*m] = 0.0;
    for ( k = 0; k < n; k++ )
    {
      bmm[i+j*m] = bmm[i+j*m] + a[i+k*m] * a_pseudo[k+j*n];
    }
  }
}
r8mat_print ( m, m, bmm, "  A * A+:" );

free ( bmm );

printf ( "\n" );
printf ( "  In some cases, the matrix A+ * A\n" );
printf ( "  may be interesting (if N <= M, then\n" );
printf ( "  it MIGHT look like the identity.)\n" );
printf ( "\n" );

bnn = ( double * ) malloc ( n * n * sizeof ( double ) );
```

```
  for ( i = 0; i < n; i++ )
  {
    for ( j = 0; j < n; j++ )
    {
      bnn[i+j*n] = 0.0;
      for ( k = 0; k < m; k++ )
      {
        bnn[i+j*n] = bnn[i+j*n] + a_pseudo[i+k*n] * a[k+j*m];
      }
    }
  }

  r8mat_print ( n, n, bnn, "  A+ * A" );

  free ( bnn );

  return;
}

int r8_nint ( double x )

{
  int s;
  int value;

  if ( x < 0.0 )
  {
    s = - 1;
  }
  else
  {
    s = + 1;
  }
  value = s * ( int ) ( fabs ( x ) + 0.5 );

  return value;
}


double r8mat_dif_fro ( int m, int n, double a[], double b[] )
{
  int i;
  int j;
  double value;

  value = 0.0;
  for ( j = 0; j < n; j++ )
  {
    for ( i = 0; i < m; i++ )
    {
      value = value + pow ( a[i+j*m] - b[i+j*m], 2 );
    }
  }
  value = sqrt ( value );

  return value;
}
```

```
double r8mat_norm_fro ( int m, int n, double a[] )

{
  int i;
  int j;
  double value;

  value = 0.0;
  for ( j = 0; j < n; j++ )
  {
    for ( i = 0; i < m; i++ )
    {
      value = value + pow ( a[i+j*m], 2 );
    }
  }
  value = sqrt ( value );

  return value;
}

void r8mat_print ( int m, int n, double a[], char *title )
{
  r8mat_print_some ( m, n, a, 1, 1, m, n, title );

  return;
}

void r8mat_print_some ( int m, int n, double a[], int ilo, int jlo, int ihi,
  int jhi, char *title )
{
# define INCX 5

  int i;
  int i2hi;
  int i2lo;
  int j;
  int j2hi;
  int j2lo;

  fprintf ( stdout, "\n" );
  fprintf ( stdout, "%s\n", title );

  if ( m <= 0 || n <= 0 )
  {
    fprintf ( stdout, "\n" );
    fprintf ( stdout, "  (None)\n" );
    return;
  }

  for ( j2lo = jlo; j2lo <= jhi; j2lo = j2lo + INCX )
  {
    j2hi = j2lo + INCX - 1;
    j2hi = i4_min ( j2hi, n );
    j2hi = i4_min ( j2hi, jhi );

    fprintf ( stdout, "\n" );
```

```
    fprintf ( stdout, "  Col:  ");
    for ( j = j2lo; j <= j2hi; j++ )
    {
      fprintf ( stdout, "  %7d     ", j - 1 );
    }
    fprintf ( stdout, "\n" );
    fprintf ( stdout, "  Row\n" );
    fprintf ( stdout, "\n" );

    i2lo = i4_max ( ilo, 1 );
    i2hi = i4_min ( ihi, m );

    for ( i = i2lo; i <= i2hi; i++ )
    {

      fprintf ( stdout, "%5d:", i - 1 );
      for ( j = j2lo; j <= j2hi; j++ )
      {
        fprintf ( stdout, "  %14f", a[i-1+(j-1)*m] );
      }
      fprintf ( stdout, "\n" );
    }
  }

  return;
# undef INCX
}


void r8mat_svd_linpack ( int m, int n, double a[], double u[], double s[],
  double v[] )
{
  double *a_copy;
  double *e;
  int i;
  int info;
  int j;
  int lda;
  int ldu;
  int ldv;
  int job;
  int lwork;
  double *sdiag;
  double *work;

  a_copy = ( double * ) malloc ( m * n * sizeof ( double ) );
  e = ( double * ) malloc ( ( m + n ) * sizeof ( double ) );
  sdiag = ( double * ) malloc ( ( m + n )  * sizeof ( double ) );
  work = ( double * ) malloc ( m * sizeof ( double ) );

  job = 11;
  lda = m;
  ldu = m;
  ldv = n;

  for ( j = 0; j < n; j++ )
```

```
  {
    for ( i = 0; i < m; i++ )
    {
      a_copy[i+j*m] = a[i+j*m];
    }
  }
  info = dsvdc ( a_copy, lda, m, n, sdiag, e, u, ldu, v, ldv, work, job );

  if ( info != 0 )
  {
    printf ( "\n" );
    printf ( "R8MAT_SVD_LINPACK - Failure!\n" );
    printf ( "  The SVD could not be calculated.\n" );
    printf ( "  LINPACK routine DSVDC returned a nonzero\n" );
    printf ( "  value of the error flag, INFO = %d\n", info );
    return;
  }

  for ( j = 0; j < n; j++ )
  {
    for ( i = 0; i < m; i++ )
    {
      if ( i == j )
      {
        s[i+j*m] = sdiag[i];
      }
      else
      {
        s[i+j*m] = 0.0;
      }
    }
  }

  free ( a_copy );
  free ( e );
  free ( sdiag );
  free ( work );

  return;
}

double *r8mat_uniform_01_new ( int m, int n, int *seed )
{
  int i;
  int j;
  int k;
  double *r;

  r = ( double * ) malloc ( m * n * sizeof ( double ) );

  for ( j = 0; j < n; j++ )
  {
    for ( i = 0; i < m; i++ )
    {
      k = *seed / 127773;

      *seed = 16807 * ( *seed - k * 127773 ) - k * 2836;
```

```
      if ( *seed < 0 )
      {
        *seed = *seed + 2147483647;
      }
      r[i+j*m] = ( double ) ( *seed ) * 4.656612875E-10;
    }
  }

  return r;
}

double r8vec_norm_l2 ( int n, double a[] )

{
  int i;
  double v;

  v = 0.0;

  for ( i = 0; i < n; i++ )
  {
    v = v + a[i] * a[i];
  }
  v = sqrt ( v );

  return v;
}


double *r8vec_uniform_01_new ( int n, int *seed )
{
  int i;
  int i4_huge = 2147483647;
  int k;
  double *r;

  if ( *seed == 0 )
  {
    fprintf ( stderr, "\n" );
    fprintf ( stderr, "R8VEC_UNIFORM_01_NEW - Fatal error!\n" );
    fprintf ( stderr, "  Input value of SEED = 0.\n" );
    exit ( 1 );
  }

  r = ( double * ) malloc ( n * sizeof ( double ) );

  for ( i = 0; i < n; i++ )
  {
    k = *seed / 127773;

    *seed = 16807 * ( *seed - k * 127773 ) - k * 2836;

    if ( *seed < 0 )
    {
      *seed = *seed + i4_huge;
    }
```

```
    r[i] = ( double ) ( *seed ) * 4.656612875E-10;
  }

  return r;
}

void rank_one_print_test ( int m, int n, double a[], double u[],
  double s[], double v[] )
{
  double a_norm;
  double dif_norm;
  int i;
  int j;
  int k;
  int r;
  double *svt;
  char title[100];
  double *usvt;

  a_norm = r8mat_norm_fro ( m, n, a );

  printf ( "\n" );
  printf ( "RANK_ONE_PRINT_TEST:\n" );
  printf ( "  Print the sums of R rank one matrices.\n" );

  for ( r = 0; r <= i4_min ( m, n ); r++ )
  {
    svt = ( double * ) malloc ( r * n * sizeof ( double ) );
    for ( i = 0; i < r; i++ )
    {
      for ( j = 0; j < n; j++ )
      {
        svt[i+j*r] = 0.0;
        for ( k = 0; k < r; k++ )
        {
          svt[i+j*r] = svt[i+j*r] + s[i+k*m] * v[j+k*n];
        }
      }
    }
    usvt = ( double * ) malloc ( m * n * sizeof ( double ) );

    for ( i = 0; i < m; i++ )
    {
      for ( j = 0; j < n; j++ )
      {
        usvt[i+j*m] = 0.0;
        for ( k = 0; k < r; k++ )
        {
          usvt[i+j*m] = usvt[i+j*m] + u[i+k*m] * svt[k+j*r];
        }
      }
    }

    sprintf ( title, "  Rank R = %d", r );
    r8mat_print ( m, n, usvt, title );
```

```
    free ( svt );
    free ( usvt );
  }

  r8mat_print ( m, n, a, "  Original matrix A:" );

  return;
}

void rank_one_test ( int m, int n, double a[], double u[], double s[],
  double v[] )
{
  double a_norm;
  double dif_norm;
  int i;
  int j;
  int k;
  int r;
  double *svt;
  double *usvt;

  a_norm = r8mat_norm_fro ( m, n, a );

  printf ( "\n" );
  printf ( "RANK_ONE_TEST:\n" );
  printf ( "  Compare A to the sum of R rank one matrices.\n" );
  printf ( "\n" );
  printf ( "         R    Absolute      Relative\n" );
  printf ( "              Error         Error\n" );
  printf ( "\n" );

  for ( r = 0; r <= i4_min ( m, n ); r++ )
  {
    svt = ( double * ) malloc ( r * n * sizeof ( double ) );
    for ( i = 0; i < r; i++ )
    {
      for ( j = 0; j < n; j++ )
      {
        svt[i+j*r] = 0.0;
        for ( k = 0; k < r; k++ )
        {
          svt[i+j*r] = svt[i+j*r] + s[i+k*m] * v[j+k*n];
        }
      }
    }
    usvt = ( double * ) malloc ( m * n * sizeof ( double ) );

    for ( i = 0; i < m; i++ )
    {
      for ( j = 0; j < n; j++ )
      {
        usvt[i+j*m] = 0.0;
        for ( k = 0; k < r; k++ )
        {
          usvt[i+j*m] = usvt[i+j*m] + u[i+k*m] * svt[k+j*r];
        }
```

```
      }
    }
    dif_norm = r8mat_dif_fro ( m, n, a, usvt );

    printf ( "  %8d  %14g  %14g\n", r, dif_norm, dif_norm / a_norm );

    free ( svt );
    free ( usvt );
  }
  return;
}

int s_len_trim ( char *s )
{
  int n;
  char *t;

  n = strlen ( s );
  t = s + strlen ( s ) - 1;

  while ( 0 < n )
  {
    if ( *t != ' ' )
    {
      return n;
    }
    t--;
    n--;
  }

  return n;
}

void svd_product_test ( int m, int n, double a[], double u[],
  double s[], double v[] )
{
  double a_norm;
  double dif_norm;
  int i;
  int j;
  int k;
  double *svt;
  double *usvt;

  a_norm = r8mat_norm_fro ( m, n, a );

  svt = ( double * ) malloc ( m * n * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
    for ( j = 0; j < n; j++ )
    {
      svt[i+j*m] = 0.0;
      for ( k = 0; k < n; k++ )
      {
        svt[i+j*m] = svt[i+j*m] + s[i+k*m] * v[j+k*n];
      }
    }
```

```
  }
  usvt = ( double * ) malloc ( m * n * sizeof ( double ) );

  for ( i = 0; i < m; i++ )
  {
    for ( j = 0; j < n; j++ )
    {
      usvt[i+j*m] = 0.0;
      for ( k = 0; k < m; k++ )
      {
        usvt[i+j*m] = usvt[i+j*m] + u[i+k*m] * svt[k+j*m];
      }
    }
  }

  r8mat_print ( m, n, usvt, "  The product U * S * V':" );

  dif_norm = r8mat_dif_fro ( m, n, a, usvt );

  printf ( "\n" );
  printf ( "  Frobenius Norm of A, A_NORM = %g\n", a_norm );
  printf ( "\n" );
  printf ( "  ABSOLUTE ERROR for A = U*S*V'\n" );
  printf ( "  Frobenius norm of difference A-U*S*V' = %g\n", dif_norm );
  printf ( "\n" );
  printf ( "  RELATIVE ERROR for A = U*S*V':\n" );
  printf ( "  Ratio of DIF_NORM / A_NORM = %g\n", dif_norm / a_norm );

  free ( svt );
  free ( usvt );

  return;
}

void timestamp ( void )
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  fprintf ( stdout, "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```

RESULT

The program has been executed successfully and the output is verified**.**

OUTPUT

```
Matrix svd.u
   0.200301   -0.593391    0.170310
   0.217730    0.768024    0.397079
   0.529515   -0.224076    0.673247
   0.795039    0.088406   -0.600051
Diagonal of matrix w (svd.w)
  26.171503    3.932220    1.609364
Matrix v-transpose (svd.v)
   0.585126    0.552922    0.593215
   0.372832    0.466199   -0.802281
  -0.720155    0.690606    0.066638

Check product against original matrix:
Original matrix:
   2.000000    2.000000    5.000000
   4.000000    5.000000    1.000000
   7.000000    8.000000    9.000000
  13.000000   11.000000   12.000000
 Product u*w*(v-transpose):
   2.000000    2.000000    5.000000
   4.000000    5.000000    1.000000
   7.000000    8.000000    9.000000
  13.000000   11.000000   12.000000
```