18 hours ago

A C-program to calculate inverse of a square matrix based on LUP decomposition

>> Click here [https://drive.google.com/file/d/0B9ydM7apiyBBQkNLQmhoRlE3M3c/view?usp=sharing] to download the C-program and its output file <<

Many of you might already be aware that calculating inverse of a square matrix **A** using the Gauss-elimination technique becomes unstable in many cases. A well-known workaround of it is to use LUP decomposition technique. In the LUP technique, first we decompose the to-be-inverted matrix as lower (**L**) and upper (**U**) triangular matrices. While doing so, a partial pivoting technique is often used, which interchanges rows (or columns) of the matrix dynamically. These interchanges are recorded in a matrix **P**, whose each row contains exactly one instance of 1 and all remaining are zeros, such that when **P** is multiplied with **A**, it is equal to the product of the decomposed triangular matrices **L** and **U**. Mathematically,

PA = LU.

In order to make the decomposition unique for a given **P**, all the diagonal elements of **L** are set to 1. For details on LUP decomposition, please look at some other resources, such as here [https://en.wikipedia.org/wiki/LU_decomposition] .

The LUP decomposition enables easy solution of a matrix equation **Ax = b**. We can re-write this equation as

PAx = Pb
=> **LUx = Pb**

which can be rewritten as the following system of two equations:

Ly = Pb (i)
and **Ux = y** (ii).

Since both the **L** and the **U** matrices are triangular, they can be easily solved by performing forward and back substitutions.

Now, imagine that **b** is a column of the identity matrix of size same as the size of **A**. We solve the equation **PAx = Pe** for each column vector **e** of the identity matrix, followed by assembling a matrix from the column vectors obtained as the solution **x**. Let us refer the solution vector corresponding to *i*-th column of the identity matrix as **x_i**. Is this assembled matrix **[x_1, x_2, ...]** the inverse of **A**?? Yes, it is, precisely because **e** are the columns of the identity matrix! This can be verified by multiplying the assembled matrix with **A**, which would yield the identity matrix (because **(A^-1)*A = I** - the identity matrix).

I implemented the above methodology in the C programming language. The program - *Matrix_inverse_LUP.c* - consists of three functions: 1. *main()*, 2. *LUPdecompose()* and 3. *LUPinverse()*. In *main()*, the matrix **A** is prepared, which then calls the other two functions in order; in *LUPdecompose()*, the LUP decomposition is performed; and in *LUPinverse()*, the decomposed values are used to solve **Ly = Pe** and **Ux = y** equations. The inverted matrix **A^-1** is then multiplied with the original matrix **A**, to confirm the

relationship **I = (A^-1)*(A)**, where **I** is the identity matrix. Ideally, the matrix **I** should contain 1's along the diagonal and 0's elsewhere. However, due to limited precision available on computers, they slightly deviate from the ideal values.

The C-program is given below. The program is well commented, and therefore it can be easily modified to make it suitable for other purposes. After the program, I will present the output of the program for three different calculations, which are performed on a same matrix **A**, but by using three data types *float*, *double*, and *__float128*, which allow low, medium, and high precision computation of the inverse matrix, respectively. The precision for these data types are up to 6, 15, and 33 decimal digits, respectively. These output would show that the deviations in the elements of **I** are largest, moderate, and least for the three data types, respectively.

The program and the output files can be downloaded together, in the zip format, from here [https://drive.google.com/file/d/0B9ydM7apiyBBQkNLQmhoRlE3M3c/view?usp=sharing] .

The program

```
/* Copyright 2015 Chandra Shekhar (chandraiitk AT yahoo DOT co DOT in).
  Homepage: https://sites.google.com/site/chandraacads
 * * */


/* This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program. If not, see <http://www.gnu.org/licenses/>.
 * * */


/* This program computes inverse of a square matrix, based on LUP decomposition.
 *
 * Tested with GCC-4.8.3 on 64 bit Linux (Fedora-20).
 *
 * Compilation:     "gcc -O2 Matrix_inverse_LUP.c -o Mat_inv_LUP.exe -lm -Wall"
 * Execution:     "./Mat_inv_LUP.exe"
 * * */


# include <stdio.h>
```

```
# include <float.h>
# include <math.h>

/* The following header file contains a 128 bit float data type '__float128',
 * used for high-precision calculations. This data-type is available only in the
 * GCC version 4.6 or later. In order to enable high-precision calculations,
 * uncomment the following line, followed by type-defining 'sankhya' appropriately
 * and including an additional '-lquadmath' flag during the compilation.
 * * */
//# include <quadmath.h>


/* The following type-define can be used to change the precision of the calculation.
 * Possible values:
 *  'float'          - for least precision
 *  'double'          - for medium precision
 *  '__float128'     - for highest precision
 * * */
typedef      double      sankhya;


# define     N     5 //Size of the to-be-inverted N x N square matrix 'A'.


/* This function performs LUP decomposition of the to-be-inverted matrix 'A'. It is
 * defined after the function 'main()'.
 * * */
static int LUPdecompose(int size, sankhya A[size][size], int P[size]);

/* This function calculates inverse of the matrix A. It accepts the LUP decomposed
 * matrix through 'LU' and the corresponding pivot through 'P'. The inverse is
 * returned through 'LU' itself. The spaces 'B', 'X', and 'Y' are used temporary,
 * merely to facilitate the computation. This function is defined after the function
 * 'LUPdecompose()'.
 * * */
static int LUPinverse(int size, int P[size], sankhya LU[size][size],\
           sankhya B[size][size], sankhya X[size], sankhya Y[size]);


int main()
 {
 int i, j, k; //Running indices.

/* 'A' is the to-be-inverted matrix. A1 is its copy, which is used to calculate
 * 'I = inverse(A).A1'. Ideally, 'I' should be a perfect identity matrix. 'I' is
 * used to check the quality of the calculated inverse. The quality increases as
 * we use 'float', 'double', and '__float128', respectively. */
 sankhya A[N+1][N+1], A1[N+1][N+1], I[N+1][N+1];
```

```
/* Its i-th row shows the position of '1' in the i-th row of the pivot that is used
 * when performing the LUP decomposition of A. The rest of the elements in that row of
 * the pivot would be zero. In this program, we call this array 'P' a 'permutation'. */
 int P[N+1];

 sankhya B[N+1][N+1], X[N+1], Y[N+1]; //Temporaty spaces.

/* Copyright notice. */
 printf("\nCopyright 2015 Chandra Shekhar (chandraiitk AT yahoo DOT co DOT in).\n"\
     "Homepage: https://sites.google.com/site/chandraacads\n\n");



/* Printing information on size and precision of various data types. */
 printf("On this machine, size (in bytes) and precision (in number of decimal digits)"\
     " of\n\tfloat: %ld and %d,\n\tdouble: %ld and %d,", sizeof(float), FLT_DIG,\
     sizeof(double), DBL_DIG);

# ifdef QUADMATH_H
 printf("\n\t__float128: %ld and %d,", sizeof(__float128), FLT128_DIG);
# endif
 printf(" respectively.");

/* Defining the to-be-inverted matrix, A. A1 would be used later to test the inverted
 * matrix. */
 for(i = 1; i <= N; i++) for(j = 1; j <= N; j++)
  A[i][j] = A1[i][j] = sin(i*j*j+i)*2;

 printf("\n\nThe to-be-inverted matrix 'A':\n");
 for(i = 1; i <= N; i++)
   {
   for(j = 1; j <= N; j++) printf("\t%E", (float)A[i][j]);
   printf("\n");
   }

/* Performing LUP-decomposition of the matrix 'A'. If successful, the 'U' is stored in
 * its upper diagonal, and the 'L' is stored in the remaining traigular space. Note that
 * all the diagonal elements of 'L' are 1, which are not stored. */
 if(LUPdecompose(N+1, A, P) < 0) return -1;
 printf("\n\nThe LUP decomposition of 'A' is successful.\nPivot:\n");
 for(i = 1; i <= N; i++)
   {
       for(j = 1; j <= N; j++) printf("\t%d", j == P[i] ? 1:0);
       printf("\n");
   }

 printf("\nLU (where 1. the diagonal of the matrix belongs to 'U', and 2. the\n"\
     "diagonal elements of 'L' are not printed, because they are all 1):\n");
 for(i = 1; i <= N; i++)
   {
```

```
   for(j = 1; j <= N; j++) printf("\t%E", (float)A[i][j]);
   printf("\n");
   }

/* Inverting the matrix based on the LUP decomposed A. The inverse is returned through
 * the matrix 'A' itself. */
  if(LUPinverse(N+1, P, A, B, X, Y) < 0) return -1;
  printf("\n\nMatrix inversion successful.\nInverse of A:\n");
  for(j = 1; j <= N; j++)
   {
    for(i = 1; i <= N; i++) printf("\t%E", (float)A[i][j]);
    printf("\n");
   }

/* Multiplying the inverse-of-A (stored in A) with A (stored in A1). The product is
 * stored in 'I'. Ideally, 'I' should be a perfect identity matrix. */
 for(i=1; i <= N; i++) for(j = 1; j <= N; j++)
  for(I[i][j] = 0, k = 1; k <= N; k++) I[i][j] += A[i][k]*A1[k][j];

 printf("\nProduct of the calculated inverse-of-A with A:\n");
 for(i = 1; i <= N; i++)
   {
   for(j = 1; j <= N; j++) printf("\t%E", (float)I[i][j]);
   printf("\n");
   }
 printf("\n");

 return 0;
 }



/* This function decomposes the matrix 'A' into L, U, and P. If successful,
 * the L and the U are stored in 'A', and information about the pivot in 'P'.
 * The diagonal elements of 'L' are all 1, and therefore they are not stored. */
static int LUPdecompose(int size, sankhya A[size][size], int P[size])
  {
  int i, j, k, kd = 0, T;
  sankhya p, t;

 /* Finding the pivot of the LUP decomposition. */
  for(i=1; i<size; i++) P[i] = i; //Initializing.

  for(k=1; k<size-1; k++)
    {
    p = 0;
    for(i=k; i<size; i++)
      {
      t = A[i][k];
```

```
      if(t < 0) t *= -1; //Abosolute value of 't'.
      if(t > p)
        {
         p = t;
         kd = i;
        }
      }

    if(p == 0)
      {
      printf("\nLUPdecompose(): ERROR: A singular matrix is supplied.\n"\
          "\tRefusing to proceed any further.\n");
      return -1;
      }

   /* Exchanging the rows according to the pivot determined above. */
    T = P[kd];
    P[kd] = P[k];
    P[k] = T;
    for(i=1; i<size; i++)
      {
       t = A[kd][i];
       A[kd][i] = A[k][i];
       A[k][i] = t;
      }

    for(i=k+1; i<size; i++) //Performing substraction to decompose A as LU.
      {
       A[i][k] = A[i][k]/A[k][k];
       for(j=k+1; j<size; j++) A[i][j] -= A[i][k]*A[k][j];
      }
    } //Now, 'A' contains the L (without the diagonal elements, which are all 1)
     //and the U.

  return 0;
  }



/* This function calculates the inverse of the LUP decomposed matrix 'LU' and pivoting
 * information stored in 'P'. The inverse is returned through the matrix 'LU' itselt.
 * 'B', X', and 'Y' are used as temporary spaces. */
static int LUPinverse(int size, int P[size], sankhya LU[size][size],\
           sankhya B[size][size], sankhya X[size], sankhya Y[size])
  {
  int i, j, n, m;
  sankhya t;

 //Initializing X and Y.
```

```
  for(n=1; n<size; n++) X[n] = Y[n] = 0;

 /* Solving LUX = Pe, in order to calculate the inverse of 'A'. Here, 'e' is a column
 * vector of the identity matrix of size 'size-1'. Solving for all 'e'. */
  for(i=1; i<size; i++)
   {
  //Storing elements of the i-th column of the identity matrix in i-th row of 'B'.
   for(j = 1; j<size; j++) B[i][j] = 0;
   B[i][i] = 1;

  //Solving Ly = Pb.
   for(n=1; n<size; n++)
    {
    t = 0;
    for(m=1; m<=n-1; m++) t += LU[n][m]*Y[m];
    Y[n] = B[i][P[n]]-t;
    }

  //Solving Ux = y.
   for(n=size-1; n>=1; n--)
    {
    t = 0;
    for(m = n+1; m < size; m++) t += LU[n][m]*X[m];
    X[n] = (Y[n]-t)/LU[n][n];
    }//Now, X contains the solution.

   for(j = 1; j<size; j++) B[i][j] = X[j]; //Copying 'X' into the same row of 'B'.
   } //Now, 'B' the transpose of the inverse of 'A'.

 /* Copying transpose of 'B' into 'LU', which would the inverse of 'A'. */
  for(i=1; i<size; i++) for(j=1; j<size; j++) LU[i][j] = B[j][i];

  return 0;
  }
```

The output

1. For *float* data type

```
Copyright 2015 Chandra Shekhar (chandraiitk AT yahoo DOT co DOT in).
Homepage: https://sites.google.com/site/chandraacads

On this machine, size (in bytes) and precision (in number of decimal digits) of
     float: 4 and 6,
     double: 8 and 15,
     __float128: 16 and 33, respectively.
```

```
The to-be-inverted matrix 'A':
     1.818595E+00     -1.917849E+00     -1.088042E+00     -1.922795E+00     1.525117E+00
     -1.513605E+00     -1.088042E+00     1.825891E+00     1.058165E+00     1.973255E+00
     -5.588310E-01     1.300576E+00     -1.976063E+00     1.340458E+00     1.027957E+00
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     -1.088042E+00     -2.647035E-01     -5.247497E-01     -3.521512E-01     -1.860212E+00


The LUP decomposition of 'A' is successful.
Pivot:
     0     0     0     1     0
     1     0     0     0     0
     0     0     1     0     0
     0     0     0     0     1
     0     1     0     0     0

LU (where 1. the diagonal of the matrix belongs to 'U', and 2. the
diagonal elements of 'L' are not printed, because they are all 1):
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     9.190780E-01     -3.595984E+00     -2.457676E+00     -2.722638E-01     2.116309E+00
     -2.824209E-01     -5.050760E-01     -2.796505E+00     6.957573E-01     1.915188E+00
     -5.498728E-01     -2.055915E-01     7.530553E-02     -1.448013E+00     -1.923044E+00
     -7.649428E-01     -8.583451E-02     -9.851134E-01     -2.392720E-01     3.089409E+00


Matrix inversion successful.
Inverse of A:
     -1.455797E-01     -3.603644E-02     -1.378691E-01     -3.837341E-01     2.018131E-01
     -5.047895E-01     1.446335E-01     1.147262E-01     -4.298742E-01     3.236865E-01
     -3.951386E-01     3.714881E-01     -2.336991E-01     -3.637230E-01     3.130355E-01
     -2.845783E-01     3.610248E-01     6.908609E-02     -5.151718E-01     1.931146E-01
     -7.747702E-01     2.043238E-01     -1.443677E-01     -7.934585E-01     7.744911E-02

Product of the calculated inverse-of-A with A:
     1.000000E+00     7.450581E-08     -1.788139E-07     2.980232E-08     1.192093E-07
     -2.980232E-08     1.000000E+00     -9.685755E-08     5.215406E-08     -2.980232E-08
     -5.960464E-08     3.725290E-09     1.000000E+00     2.980232E-08     5.960464E-08
     -5.960464E-08     -1.490116E-08     -2.980232E-08     1.000000E+00     1.192093E-07
     7.450581E-09     8.381903E-08     -2.980232E-08     1.862645E-09     9.999999E-01
```

2. For *double* data type

```
Copyright 2015 Chandra Shekhar (chandraiitk AT yahoo DOT co DOT in).
Homepage: https://sites.google.com/site/chandraacads
```

```
On this machine, size (in bytes) and precision (in number of decimal digits) of
     float: 4 and 6,
     double: 8 and 15,
     __float128: 16 and 33, respectively.

The to-be-inverted matrix 'A':
     1.818595E+00     -1.917849E+00     -1.088042E+00     -1.922795E+00     1.525117E+00
     -1.513605E+00     -1.088042E+00     1.825891E+00     1.058165E+00     1.973255E+00
     -5.588310E-01     1.300576E+00     -1.976063E+00     1.340458E+00     1.027957E+00
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     -1.088042E+00     -2.647035E-01     -5.247497E-01     -3.521512E-01     -1.860212E+00


The LUP decomposition of 'A' is successful.
Pivot:
     0     0     0     1     0
     1     0     0     0     0
     0     0     1     0     0
     0     0     0     0     1
     0     1     0     0     0

LU (where 1. the diagonal of the matrix belongs to 'U', and 2. the
diagonal elements of 'L' are not printed, because they are all 1):
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     9.190781E-01     -3.595984E+00     -2.457676E+00     -2.722638E-01     2.116309E+00
     -2.824210E-01     -5.050760E-01     -2.796505E+00     6.957573E-01     1.915188E+00
     -5.498727E-01     -2.055915E-01     7.530553E-02     -1.448013E+00     -1.923044E+00
     -7.649428E-01     -8.583453E-02     -9.851134E-01     -2.392720E-01     3.089409E+00


Matrix inversion successful.
Inverse of A:
     -1.455798E-01     -3.603643E-02     -1.378691E-01     -3.837342E-01     2.018131E-01
     -5.047895E-01     1.446336E-01     1.147262E-01     -4.298742E-01     3.236865E-01
     -3.951387E-01     3.714881E-01     -2.336992E-01     -3.637230E-01     3.130355E-01
     -2.845783E-01     3.610248E-01     6.908608E-02     -5.151718E-01     1.931146E-01
     -7.747702E-01     2.043238E-01     -1.443677E-01     -7.934586E-01     7.744911E-02

Product of the calculated inverse-of-A with A:
     1.000000E+00     -2.775558E-17     -2.220446E-16     1.110223E-16     0.000000E+00
     -1.387779E-16     1.000000E+00     1.249001E-16     2.775558E-17     5.551115E-17
     -8.326673E-17     -7.632783E-17     1.000000E+00     2.775558E-17     5.551115E-17
     1.110223E-16     -3.885781E-16     -5.551115E-17     1.000000E+00     0.000000E+00
     -1.249001E-16     1.075529E-16     2.775558E-17     3.469447E-17     1.000000E+00
```

3. For *__float128* data type

```
Copyright 2015 Chandra Shekhar (chandraiitk AT yahoo DOT co DOT in).
Homepage: https://sites.google.com/site/chandraacads

On this machine, size (in bytes) and precision (in number of decimal digits) of
     float: 4 and 6,
     double: 8 and 15,
     __float128: 16 and 33, respectively.

The to-be-inverted matrix 'A':
     1.818595E+00     -1.917849E+00     -1.088042E+00     -1.922795E+00     1.525117E+00
     -1.513605E+00     -1.088042E+00     1.825891E+00     1.058165E+00     1.973255E+00
     -5.588310E-01     1.300576E+00     -1.976063E+00     1.340458E+00     1.027957E+00
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     -1.088042E+00     -2.647035E-01     -5.247497E-01     -3.521512E-01     -1.860212E+00


The LUP decomposition of 'A' is successful.
Pivot:
     0     0     0     1     0
     1     0     0     0     0
     0     0     1     0     0
     0     0     0     0     1
     0     1     0     0     0

LU (where 1. the diagonal of the matrix belongs to 'U', and 2. the
diagonal elements of 'L' are not printed, because they are all 1):
     1.978716E+00     1.825891E+00     1.490226E+00     -1.795855E+00     -6.432448E-01
     9.190781E-01     -3.595984E+00     -2.457676E+00     -2.722638E-01     2.116309E+00
     -2.824210E-01     -5.050760E-01     -2.796505E+00     6.957573E-01     1.915188E+00
     -5.498727E-01     -2.055915E-01     7.530553E-02     -1.448013E+00     -1.923044E+00
     -7.649428E-01     -8.583453E-02     -9.851134E-01     -2.392720E-01     3.089409E+00


Matrix inversion successful.
Inverse of A:
     -1.455798E-01     -3.603643E-02     -1.378691E-01     -3.837342E-01     2.018131E-01
     -5.047895E-01     1.446336E-01     1.147262E-01     -4.298742E-01     3.236865E-01
     -3.951387E-01     3.714881E-01     -2.336992E-01     -3.637230E-01     3.130355E-01
     -2.845783E-01     3.610248E-01     6.908608E-02     -5.151718E-01     1.931146E-01
     -7.747702E-01     2.043238E-01     -1.443677E-01     -7.934586E-01     7.744911E-02

Product of the calculated inverse-of-A with A:
     1.000000E+00     -9.629650E-35     -4.814825E-35     -1.444447E-34     -1.925930E-34
     -4.814825E-35     1.000000E+00     9.629650E-35     2.407412E-35     -4.814825E-35
     0.000000E+00     -1.805559E-35     1.000000E+00     1.805559E-35     -4.814825E-35
     0.000000E+00     7.222237E-35     -1.444447E-34     1.000000E+00     -1.925930E-34
```

```
-4.814825E-35     -5.416678E-35     1.203706E-35     1.023150E-34     1.000000E+00
```

I hope it helps! If you encounter any bug in this program, please comment below or contact me here [http://spreadsheets.google.com/viewform?formkey=dElQTHFZaFhZdG9oUjJJb29HbzVUU1E6MA] . Alternatively, you can directly e-mail me also.

चन्द्र शेखर (*Chandra Shekhar*)
१३ दिसंबर, २०१५

chandraiitk@YAHOO.co.in

Posted 18 hours ago by Chandra

Labels: C program, LUP decomposition, Matrix inverse

0 Add a comment

Enter your comment...

Comment as: Google Accour ▼

Publish Preview