Cholesky decomposition

From Rosetta Code

Cholesky decomposition
You are encouraged to solve this task according to the task description, using any language you may know.

Every symmetric, positive definite matrix A can be decomposed into a product of a unique lower triangular matrix L and its transpose:

$$A = LL^T$$

L is called the *Cholesky factor* of *A*, and can be interpreted as a generalized square root of *A*, as described in Cholesky decomposition.

In a 3x3 example, we have to solve the following system of equations:

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \\ &= \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{pmatrix} \equiv LL^T \\ &= \begin{pmatrix} l_{11}^2 & l_{21}l_{11} & l_{31}l_{11} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & l_{31}l_{21} + l_{32}l_{22} \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{pmatrix} \end{aligned}$$

We can see that for the diagonal elements (l_{kk}) of L there is a calculation pattern:

$$\begin{aligned} l_{11} &= \sqrt{a_{11}} \\ l_{22} &= \sqrt{a_{22} - l_{21}^2} \\ l_{33} &= \sqrt{a_{33} - (l_{31}^2 + l_{32}^2)} \end{aligned}$$

or in general:

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

For the elements below the diagonal (l_{ik}, where $i > k$) there is also a calculation pattern:

$$\begin{aligned} l_{21} &= \frac{1}{l_{11}} a_{21} \\ l_{31} &= \frac{1}{l_{11}} a_{31} \\ l_{32} &= \frac{1}{l_{22}} (a_{32} - l_{31}l_{21}) \end{aligned}$$

which can also be expressed in a general formula:

$$l_{ik} = \frac{1}{l_{kk}} \left(a_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj} \right)$$

Task description

The task is to implement a routine which will return a lower Cholesky factor L for every given symmetric, positive definite nxn matrix A. You should then test it on the following two examples and include your output.

Example 1:

```
25  15  -5                 5   0   0
15  18   0         -->     3   3   0
-5   0  11                -1   1   3
```

Example 2:

```
18  22   54   42           4.24264    0.00000    0.00000    0.00000
22  70   86   62   -->     5.18545    6.56591    0.00000    0.00000
54  86  174  134          12.72792    3.04604    1.64974    0.00000
42  62  134  106           9.89949    1.62455    1.84971    1.39262
```

Note

1. The Cholesky decomposition of a Pascal upper-triangle matrix is the Identity matrix of the same size.
2. The Cholesky decomposition of a Pascal symmetric matrix is the Pascal lower-triangle matrix of the same size.

Contents

- 1 Ada
- 2 ALGOL 68
- 3 BBC BASIC
- 4 C
- 5 Common Lisp
- 6 D
- 7 DWScript
- 8 Fantom
- 9 Fortran
- 10 Go
  - 10.1 Real
  - 10.2 Hermitian
  - 10.3 Library
- 11 Haskell
- 12 Icon and Unicon
- 13 J
- 14 Java
- 15 jq
- 16 Julia
- 17 Maple
- 18 Mathematica
- 19 MATLAB / Octave
- 20 Maxima
- 21 Nim
- 22 Objeck
- 23 OCaml
- 24 Pascal
- 25 Perl
- 26 Perl 6
- 27 PicoLisp
- 28 PL/I
- 29 Python
  - 29.1 Python2.X version
  - 29.2 Python3.X version using extra Python idioms
- 30 q

- 31 R
- 32 Racket
- 33 REXX
- 34 Ruby
- 35 Scala
- 36 Seed7
- 37 Sidef
- 38 Smalltalk
- 39 Tcl
- 40 VBA
- 41 zkl

Ada

Works with: Ada 2005

decomposition.ads:

```
with Ada.Numerics.Generic_Real_Arrays;
generic
   with package Matrix is new Ada.Numerics.Generic_Real_Arrays (<>);
package Decomposition is

   -- decompose a square matrix A by A = L * Transpose (L)
   procedure Decompose (A : Matrix.Real_Matrix; L : out Matrix.Real_Matrix);

end Decomposition;
```

decomposition.adb:

```
with Ada.Numerics.Generic_Elementary_Functions;

package body Decomposition is
   package Math is new Ada.Numerics.Generic_Elementary_Functions
     (Matrix.Real);

   procedure Decompose (A : Matrix.Real_Matrix; L : out Matrix.Real_Matrix) is
      use type Matrix.Real_Matrix, Matrix.Real;
      Order : constant Positive := A'Length (1);
      S     : Matrix.Real;
   begin
      L := (others => (others => 0.0));
      for I in 0 .. Order - 1 loop
         for K in 0 .. I loop
            S := 0.0;
            for J in 0 .. K - 1 loop
               S := S +
                 L (L'First (1) + I, L'First (2) + J) *
                 L (L'First (1) + K, L'First (2) + J);
            end loop;
            -- diagonals
            if K = I then
               L (L'First (1) + K, L'First (2) + K) :=
                 Math.Sqrt (A (A'First (1) + K, A'First (2) + K) - S);
            else
               L (L'First (1) + I, L'First (2) + K) :=
                 1.0 / L (L'First (1) + K, L'First (2) + K) *
                 (A (A'First (1) + I, A'First (2) + K) - S);
            end if;
         end loop;
      end loop;
   end Decompose;
end Decomposition;
```

Example usage:

```
with Ada.Numerics.Real_Arrays;
with Ada.Text_IO;
with Decomposition;
procedure Decompose_Example is
   package Real_Decomposition is new Decomposition
     (Matrix => Ada.Numerics.Real_Arrays);
```

```
   package Real_IO is new Ada.Text_IO.Float_IO (Float);

   procedure Print (M : Ada.Numerics.Real_Arrays.Real_Matrix) is
   begin
      for Row in M'Range (1) loop
         for Col in M'Range (2) loop
            Real_IO.Put (M (Row, Col), 4, 3, 0);
         end loop;
         Ada.Text_IO.New_Line;
      end loop;
   end Print;

   Example_1 : constant Ada.Numerics.Real_Arrays.Real_Matrix :=
     ((25.0, 15.0, -5.0),
      (15.0, 18.0, 0.0),
      (-5.0, 0.0, 11.0));
   L_1 : Ada.Numerics.Real_Arrays.Real_Matrix (Example_1'Range (1),
                                               Example_1'Range (2));
   Example_2 : constant Ada.Numerics.Real_Arrays.Real_Matrix :=
     ((18.0, 22.0, 54.0, 42.0),
      (22.0, 70.0, 86.0, 62.0),
      (54.0, 86.0, 174.0, 134.0),
      (42.0, 62.0, 134.0, 106.0));
   L_2 : Ada.Numerics.Real_Arrays.Real_Matrix (Example_2'Range (1),
                                               Example_2'Range (2));
begin
   Real_Decomposition.Decompose (A => Example_1,
                                 L => L_1);
   Real_Decomposition.Decompose (A => Example_2,
                                 L => L_2);
   Ada.Text_IO.Put_Line ("Example 1:");
   Ada.Text_IO.Put_Line ("A:"); Print (Example_1);
   Ada.Text_IO.Put_Line ("L:"); Print (L_1);
   Ada.Text_IO.New_Line;
   Ada.Text_IO.Put_Line ("Example 2:");
   Ada.Text_IO.Put_Line ("A:"); Print (Example_2);
   Ada.Text_IO.Put_Line ("L:"); Print (L_2);
end Decompose_Example;
```

Output:

```
Example 1:
A:
  25.000  15.000  -5.000
  15.000  18.000   0.000
  -5.000   0.000  11.000
L:
   5.000   0.000   0.000
   3.000   3.000   0.000
  -1.000   1.000   3.000

Example 2:
A:
  18.000  22.000  54.000  42.000
  22.000  70.000  86.000  62.000
  54.000  86.000 174.000 134.000
  42.000  62.000 134.000 106.000
L:
   4.243   0.000   0.000   0.000
   5.185   6.566   0.000   0.000
  12.728   3.046   1.650   0.000
   9.899   1.625   1.850   1.393
```

ALGOL 68

Translation of: C
Note: This specimen retains the original C coding style. diff (http://rosettacode.org/mw/index.php?title=Cholesky_decomposition&action=historysubmit&diff=107753&oldid=107752)
Works with: ALGOL 68 version Revision 1 - no extensions to language used.
Works with: ALGOL 68G version Any - tested with release 1.18.0-9h.tiny (http://sourceforge.net/projects/algol68/files/algol68g/algol68g-1.18.0/algol68g-1.18.0-9h.tiny.el5.centos.fc11.i386.rpm/download) .

```
#!/usr/local/bin/a68g --script #

MODE FIELD=LONG REAL;
PROC (FIELD)FIELD field sqrt = long sqrt;
INT field prec = 5;
FORMAT field fmt = $g(-(2+1+field prec),field prec)$;

MODE MAT = [0,0]FIELD;
```

```
PROC cholesky = (MAT a) MAT:(
    [UPB a, 2 UPB a]FIELD l;

    FOR i FROM LWB a TO UPB a DO
        FOR j FROM 2 LWB a TO i DO
            FIELD s := 0;
            FOR k FROM 2 LWB a TO j-1 DO
                s +:= l[i,k] * l[j,k]
            OD;
            l[i,j] := IF i = j
                      THEN field sqrt(a[i,i] - s)
                      ELSE 1.0 / l[j,j] * (a[i,j] - s) FI
        OD;
        FOR j FROM i+1 TO 2 UPB a DO
            l[i,j]:=0 # Not required if matrix is declared as triangular #
        OD
    OD;
    l
);

PROC print matrix v1 =(MAT a)VOID:(
    FOR i FROM LWB a TO UPB a DO
        FOR j FROM 2 LWB a TO 2 UPB a DO
            printf(($g(-(2+1+field prec),field prec)$, a[i,j]))
        OD;
        printf($l$)
    OD
);

PROC print matrix =(MAT a)VOID:(
    FORMAT vector fmt = $"("f(field  fmt)n(2 UPB a-2 LWB a)(", " f(field  fmt))")"$;
    FORMAT matrix fmt = $"("f(vector fmt)n(  UPB a-  LWB a)(","lxf(vector fmt))")"$;
    printf((matrix fmt, a))
);

main: (
    MAT m1 = ((25, 15, -5),
              (15, 18,  0),
              (-5,  0, 11));
    MAT c1 = cholesky(m1);
    print matrix(c1);
    printf($l$);

    MAT m2 = ((18, 22,  54,  42),
              (22, 70,  86,  62),
              (54, 86, 174, 134),
              (42, 62, 134, 106));
    MAT c2 = cholesky(m2);
    print matrix(c2)
)
```

Output:

```
(( 5.00000,  0.00000,  0.00000),
 ( 3.00000,  3.00000,  0.00000),
 (-1.00000,  1.00000,  3.00000))
(( 4.24264,  0.00000,  0.00000,  0.00000),
 ( 5.18545,  6.56591,  0.00000,  0.00000),
 (12.72792,  3.04604,  1.64974,  0.00000),
 ( 9.89949,  1.62455,  1.84971,  1.39262))
```

BBC BASIC

Works with: BBC BASIC for Windows

```
      DIM m1(2,2)
      m1() = 25, 15, -5, \
      \      15, 18,  0, \
      \      -5,  0, 11
      PROCcholesky(m1())
      PROCprint(m1())
      PRINT

      @% = &2050A
      DIM m2(3,3)
      m2() = 18, 22,  54,  42, \
      \      22, 70,  86,  62, \
      \      54, 86, 174, 134, \
      \      42, 62, 134, 106
      PROCcholesky(m2())
      PROCprint(m2())
      END
```

```
      DEF PROCcholesky(a())
      LOCAL i%, j%, k%, l(), s
      DIM l(DIM(a(),1),DIM(a(),2))
      FOR i% = 0 TO DIM(a(),1)
        FOR j% = 0 TO i%
          s = 0
          FOR k% = 0 TO j%-1
            s += l(i%,k%) * l(j%,k%)
          NEXT
          IF i% = j% THEN
            l(i%,j%) = SQR(a(i%,i%) - s)
          ELSE
            l(i%,j%) = (a(i%,j%) - s) / l(j%,j%)
          ENDIF
        NEXT j%
      NEXT i%
      a() = l()
      ENDPROC

      DEF PROCprint(a())
      LOCAL row%, col%
      FOR row% = 0 TO DIM(a(),1)
        FOR col% = 0 TO DIM(a(),2)
          PRINT a(row%,col%);
        NEXT
        PRINT
      NEXT row%
      ENDPROC
```

Output:

```
         5         0         0
         3         3         0
        -1         1         3

   4.24264   0.00000   0.00000   0.00000
   5.18545   6.56591   0.00000   0.00000
  12.72792   3.04604   1.64974   0.00000
   9.89949   1.62455   1.84971   1.39262
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double *cholesky(double *A, int n) {
    double *L = (double*)calloc(n * n, sizeof(double));
    if (L == NULL)
        exit(EXIT_FAILURE);

    for (int i = 0; i < n; i++)
        for (int j = 0; j < (i+1); j++) {
            double s = 0;
            for (int k = 0; k < j; k++)
                s += L[i * n + k] * L[j * n + k];
            L[i * n + j] = (i == j) ?
                           sqrt(A[i * n + i] - s) :
                           (1.0 / L[j * n + j] * (A[i * n + j] - s));
        }

    return L;
}

void show_matrix(double *A, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%2.5f ", A[i * n + j]);
        printf("\n");
    }
}

int main() {
    int n = 3;
    double m1[] = {25, 15, -5,
                   15, 18,  0,
                   -5,  0, 11};
    double *c1 = cholesky(m1, n);
    show_matrix(c1, n);
    printf("\n");
    free(c1);

    n = 4;
```

```
    double m2[] = {18, 22,  54,  42,
                   22, 70,  86,  62,
                   54, 86, 174, 134,
                   42, 62, 134, 106};
    double *c2 = cholesky(m2, n);
    show_matrix(c2, n);
    free(c2);

    return 0;
}
```

Output:

```
5.00000 0.00000 0.00000
3.00000 3.00000 0.00000
-1.00000 1.00000 3.00000

4.24264 0.00000 0.00000 0.00000
5.18545 6.56591 0.00000 0.00000
12.72792 3.04604 1.64974 0.00000
9.89949 1.62455 1.84971 1.39262
```

Common Lisp

```
;; Calculates the Cholesky decomposition matrix L
;; for a positive-definite, symmetric nxn matrix A.
(defun chol (A)
  (let* ((n (car (array-dimensions A)))
         (L (make-array `(,n ,n) :initial-element 0)))

    (do ((k 0 (incf k))) ((> k (- n 1)) nil)
        ;; First, calculate diagonal elements L_kk.
        (setf (aref L k k)
              (sqrt (- (aref A k k)
                       (do* ((j 0 (incf j))
                             (sum (expt (aref L k j) 2)
                                  (incf sum (expt (aref L k j) 2))))
                            ((> j (- k 1)) sum)))))

        ;; Then, all elements below a diagonal element, L_ik, i=k+1..n.
        (do ((i (+ k 1) (incf i)))
            ((> i (- n 1)) nil)

            (setf (aref L i k)
                  (/ (- (aref A i k)
                        (do* ((j 0 (incf j))
                              (sum (* (aref L i j) (aref L k j))
                                   (incf sum (* (aref L i j) (aref L k j)))))
                             ((> j (- k 1)) sum)))
                     (aref L k k)))))

    ;; Return the calculated matrix L.
    L))
```

```
;; Example 1:
(setf A (make-array '(3 3) :initial-contents '((25 15 -5) (15 18 0) (-5 0 11))))
(chol A)
#2A((5.0 0 0)
    (3.0 3.0 0)
    (-1.0 1.0 3.0))
```

```
;; Example 2:
(setf B (make-array '(4 4) :initial-contents '((18 22 54 42) (22 70 86 62) (54 86 174 134) (42 62 134 106))))
(chol B)
#2A((4.2426405 0 0 0)
    (5.18545 6.565905 0 0)
    (12.727922 3.0460374 1.6497375 0)
    (9.899495 1.6245536 1.849715 1.3926151))
```

```
;; case of matrix stored as a list of lists (inner lists are rows of matrix)
;; as above, returns the Cholesky decomposition matrix of a square positive-definite, symmetric matrix
(defun cholesky (m)
  (let ((l (list (list (sqrt (caar m))))) x (j 0) i)
    (dolist (cm (cdr m) (mapcar #'(lambda (x) (nconc x (make-list (- (length m) (length x)) :initial-element 0))) l))
      (setq x (list (/ (car cm) (caar l))) i 0)
      (dolist (cl (cdr l))
        (setf (cdr (last x)) (list (/ (- (elt cm (incf i)) (*v x cl)) (car (last cl))))))
      (setf (cdr (last l)) (list (nconc x (list (sqrt (- (elt cm (incf j)) (*v x x))))))))))
```

```
;; where *v is the scalar product defined as
(defun *v (v1 v2) (reduce #'+ (mapcar #'* v1 v2)))
```

```
;; example 1
CL-USER> (setf a '((25 15 -5) (15 18 0) (-5 0 11)))
((25 15 -5) (15 18 0) (-5 0 11))
CL-USER> (cholesky a)
((5 0 0) (3 3 0) (-1 1 3))
CL-USER> (format t "~{~{~5d~}~%~}" (cholesky a))
    5    0    0
    3    3    0
   -1    1    3
NIL
```

```
;; example 2
CL-USER> (setf a '((18 22 54 42) (22 70 86 62) (54 86 174 134) (42 62 134 106)))
((18 22 54 42) (22 70 86 62) (54 86 174 134) (42 62 134 106))
CL-USER> (cholesky a)
((4.2426405 0 0 0) (5.18545 6.565905 0 0) (12.727922 3.0460374 1.6497375 0) (9.899495 1.6245536 1.849715 1.3926151))
CL-USER> (format t "~{~{~10,5f~}~%~}" (cholesky a))
   4.24264   0.00000   0.00000   0.00000
   5.18545   6.56591   0.00000   0.00000
  12.72792   3.04604   1.64974   0.00000
   9.89950   1.62455   1.84971   1.39262
NIL
```

D

```
import std.stdio, std.math, std.numeric;

T[][] cholesky(T)(in T[][] A) pure nothrow /*@safe*/ {
    auto L = new T[][](A.length, A.length);
    foreach (immutable r, row; L)
        row[r + 1 .. $] = 0;
    foreach (immutable i; 0 .. A.length)
        foreach (immutable j; 0 .. i + 1) {
            auto t = dotProduct(L[i][0 .. j], L[j][0 .. j]);
            L[i][j] = (i == j) ? (A[i][i] - t) ^^ 0.5 :
                                 (1.0 / L[j][j] * (A[i][j] - t));
        }
    return L;
}

void main() {
    immutable double[][] m1 = [[25, 15, -5],
                               [15, 18,  0],
                               [-5,  0, 11]];
    writefln("%(%(%2.0f %)\n%)\n", m1.cholesky);

    immutable double[][] m2 = [[18, 22,  54,  42],
                               [22, 70,  86,  62],
                               [54, 86, 174, 134],
                               [42, 62, 134, 106]];
    writefln("%(%(%2.3f %)\n%)", m2.cholesky);
}
```

Output:

```
 5  0  0
 3  3  0
-1  1  3

4.243 0.000 0.000 0.000
5.185 6.566 0.000 0.000
12.728 3.046 1.650 0.000
9.899 1.625 1.850 1.393
```

DWScript

Translation of: C

```
function Cholesky(a : array of Float) : array of Float;
var
   i, j, k, n : Integer;
   s : Float;
```

```
begin
   n:=Round(Sqrt(a.Length));
   Result:=new Float[n*n];
   for i:=0 to n-1 do begin
      for j:=0 to i do begin
         s:=0 ;
         for k:=0 to j-1 do
            s+=Result[i*n+k] * Result[j*n+k];
         if i=j then
            Result[i*n+j]:=Sqrt(a[i*n+i]-s)
         else Result[i*n+j]:=1/Result[j*n+j]*(a[i*n+j]-s);
      end;
   end;
end;

procedure ShowMatrix(a : array of Float);
var
   i, j, n : Integer;
begin
   n:=Round(Sqrt(a.Length));
   for i:=0 to n-1 do begin
      for j:=0 to n-1 do
         Print(Format('%2.5f ', [a[i*n+j]]));
      PrintLn('');
   end;
end;

var m1 := new Float[9];
m1 := [ 25.0, 15.0, -5.0,
        15.0, 18.0,  0.0,
        -5.0,  0.0, 11.0 ];
var c1 := Cholesky(m1);
ShowMatrix(c1);

PrintLn('');

var m2 : array of Float := [ 18.0, 22.0,  54.0,  42.0,
                             22.0, 70.0,  86.0,  62.0,
                             54.0, 86.0, 174.0, 134.0,
                             42.0, 62.0, 134.0, 106.0 ];
var c2 := Cholesky(m2);
ShowMatrix(c2);
```

Fantom

```
**
** Cholesky decomposition
**

class Main
{
  // create an array of Floats, initialised to 0.0
  Float[][] makeArray (Int i, Int j)
  {
    Float[][] result := [,]
    i.times { result.add ([,]) }
    i.times |Int x|
    {
      j.times
      {
        result[x].add(0f)
      }
    }
    return result
  }

  // perform the Cholesky decomposition
  Float[][] cholesky (Float[][] array)
  {
    m := array.size
    Float[][] l := makeArray (m, m)
    m.times |Int i|
    {
      (i+1).times |Int k|
      {
        Float sum := (0..<k).toList.reduce (0f) |Float a, Int j -> Float|
        {
          a + l[i][j] * l[k][j]
        }
        if (i == k)
          l[i][k] = (array[i][i]-sum).sqrt
        else
          l[i][k] = (1.0f / l[k][k]) * (array[i][k] - sum)
      }
    }
```

```
    return l
  }

  Void runTest (Float[][] array)
  {
    echo (array)
    echo (cholesky (array))
  }

  Void main ()
  {
    runTest ([[25f,15f,-5f],[15f,18f,0f],[-5f,0f,11f]])
    runTest ([[18f,22f,54f,42f],[22f,70f,86f,62f],[54f,86f,174f,134f],[42f,62f,134f,106f]])
  }
}
```

Output:

```
[[25.0, 15.0, -5.0], [15.0, 18.0, 0.0], [-5.0, 0.0, 11.0]]
[[5.0, 0.0, 0.0], [3.0, 3.0, 0.0], [-1.0, 1.0, 3.0]]
[[18.0, 22.0, 54.0, 42.0], [22.0, 70.0, 86.0, 62.0], [54.0, 86.0, 174.0, 134.0], [42.0, 62.0, 134.0, 106.0]]
[[4.242640687119285, 0.0, 0.0, 0.0], [5.185449728701349, 6.565905201197403, 0.0, 0.0], [12.727922061357857, 3.0460384954008553, 1.6497422
```

Fortran

```
Program Cholesky_decomp
! *************************************************!
! LBH @ ULPGC 06/03/2014
! Compute the Cholesky decomposition for a matrix A
! after the attached
! http://rosettacode.org/wiki/Cholesky_decomposition
! note that the matrix A is complex since there might
! be values, where the sqrt has complex solutions.
! Here, only the real values are taken into account
!*************************************************!
implicit none

INTEGER, PARAMETER :: m=3 !rows
INTEGER, PARAMETER :: n=3 !cols
COMPLEX, DIMENSION(m,n) :: A
REAL, DIMENSION(m,n) :: L
REAL :: sum1, sum2
INTEGER i,j,k

! Assign values to the matrix
A(1,:)=(/ 25,  15,  -5 /)
A(2,:)=(/ 15,  18,   0 /)
A(3,:)=(/ -5,   0,  11 /)
! !!!!!!!!!!!another example!!!!!!!
! A(1,:) = (/ 18,  22,   54,   42 /)
! A(2,:) = (/ 22,  70,   86,   62 /)
! A(3,:) = (/ 54,  86,  174,  134 /)
! A(4,:) = (/ 42,  62,  134,  106 /)




! Initialize values
L(1,1)=real(sqrt(A(1,1)))
L(2,1)=A(2,1)/L(1,1)
L(2,2)=real(sqrt(A(2,2)-L(2,1)*L(2,1)))
L(3,1)=A(3,1)/L(1,1)
! for greater order than m,n=3 add initial row value
! for instance if m,n=4 then add the following line
! L(4,1)=A(4,1)/L(1,1)




do i=1,n
    do k=1,i
        sum1=0
        sum2=0
        do j=1,k-1
        if (i==k) then
            sum1=sum1+(L(k,j)*L(k,j))
            L(k,k)=real(sqrt(A(k,k)-sum1))
```

```
        elseif (i > k) then
            sum2=sum2+(L(i,j)*L(k,j))
            L(i,k)=(1/L(k,k))*(A(i,k)-sum2)
        else
            L(i,k)=0
        end if
        end do
    end do
end do

! write output
do i=1,m
    print "(3(1X,F6.1))",L(i,:)
end do

End program Cholesky_decomp
```

Output:

```
   5.0   0.0   0.0
   3.0   3.0   0.0
  -1.0   1.0   3.0
```

Go

Real

This version works with real matrices, like most other solutions on the page. The representation is packed, however, storing only the lower triange of the input symetric matrix and the output lower matrix. The decomposition algorithm computes rows in order from top to bottom but is a little different thatn Cholesky–Banachiewicz.

```
package main

import (
    "fmt"
    "math"
)

// symmetric and lower use a packed representation that stores only
// the lower triangle.

type symmetric struct {
    order int
    ele   []float64
}

type lower struct {
    order int
    ele   []float64
}

// symmetric.print prints a square matrix from the packed representation,
// printing the upper triange as a transpose of the lower.
func (s *symmetric) print() {
    const eleFmt = "%10.5f "
    row, diag := 1, 0
    for i, e := range s.ele {
        fmt.Printf(eleFmt, e)
        if i == diag {
            for j, col := diag+row, row; col < s.order; j += col {
                fmt.Printf(eleFmt, s.ele[j])
                col++
            }
            fmt.Println()
            row++
            diag += row
        }
    }
}

// lower.print prints a square matrix from the packed representation,
// printing the upper triangle as all zeros.
func (l *lower) print() {
    const eleFmt = "%10.5f "
    row, diag := 1, 0
    for i, e := range l.ele {
        fmt.Printf(eleFmt, e)
        if i == diag {
            for j := row; j < l.order; j++ {
                fmt.Printf(eleFmt, 0.)
```

```
            }
            fmt.Println()
            row++
            diag += row
        }
    }
}

// choleskyLower returns the cholesky decomposition of a symmetric real
// matrix.  The matrix must be positive definite but this is not checked.
func (a *symmetric) choleskyLower() *lower {
    l := &lower{a.order, make([]float64, len(a.ele))}
    row, col := 1, 1
    dr := 0 // index of diagonal element at end of row
    dc := 0 // index of diagonal element at top of column
    for i, e := range a.ele {
        if i < dr {
            d := (e - l.ele[i]) / l.ele[dc]
            l.ele[i] = d
            ci, cx := col, dc
            for j := i + 1; j <= dr; j++ {
                cx += ci
                ci++
                l.ele[j] += d * l.ele[cx]
            }
            col++
            dc += col
        } else {
            l.ele[i] = math.Sqrt(e - l.ele[i])
            row++
            dr += row
            col = 1
            dc = 0
        }
    }
    return l
}

func main() {
    demo(&symmetric{3, []float64{
        25,
        15, 18,
        -5, 0, 11}})
    demo(&symmetric{4, []float64{
        18,
        22, 70,
        54, 86, 174,
        42, 62, 134, 106}})
}

func demo(a *symmetric) {
    fmt.Println("A:")
    a.print()
    fmt.Println("L:")
    a.choleskyLower().print()
}
```

Output:

```
A:
  25.00000   15.00000   -5.00000
  15.00000   18.00000    0.00000
  -5.00000    0.00000   11.00000
L:
   5.00000    0.00000    0.00000
   3.00000    3.00000    0.00000
  -1.00000    1.00000    3.00000
A:
  18.00000   22.00000   54.00000   42.00000
  22.00000   70.00000   86.00000   62.00000
  54.00000   86.00000  174.00000  134.00000
  42.00000   62.00000  134.00000  106.00000
L:
   4.24264    0.00000    0.00000    0.00000
   5.18545    6.56591    0.00000    0.00000
  12.72792    3.04604    1.64974    0.00000
   9.89949    1.62455    1.84971    1.39262
```

Hermitian

This version handles complex Hermitian matricies as described on the WP page. The matrix representation is flat, and storage is allocated for all elements, not just the lower triangles. The decomposition algorithm is Cholesky–Banachiewicz.

```
package main

import (
    "fmt"
    "math/cmplx"
)

type matrix struct {
    ele    []complex128
    stride int
}

func matrixFromRows(rows [][]complex128) *matrix {
    if len(rows) == 0 {
        return &matrix{nil, 0}
    }
    m := &matrix{make([]complex128, len(rows)*len(rows[0])), len(rows[0])}
    for rx, row := range rows {
        copy(m.ele[rx*m.stride:(rx+1)*m.stride], row)
    }
    return m
}

func like(a *matrix) *matrix {
    return &matrix{make([]complex128, len(a.ele)), a.stride}
}

func (m *matrix) print(heading string) {
    if heading > "" {
        fmt.Print("\n", heading, "\n")
    }
    for e := 0; e < len(m.ele); e += m.stride {
        fmt.Printf("%7.2f ", m.ele[e:e+m.stride])
        fmt.Println()
    }
}

func (a *matrix) choleskyDecomp() *matrix {
    l := like(a)
    // Cholesky-Banachiewicz algorithm
    for r, rxc0 := 0, 0; r < a.stride; r++ {
        // calculate elements along row, up to diagonal
        x := rxc0
        for c, cxc0 := 0, 0; c < r; c++ {
            sum := a.ele[x]
            for k := 0; k < c; k++ {
                sum -= l.ele[rxc0+k] * cmplx.Conj(l.ele[cxc0+k])
            }
            l.ele[x] = sum / l.ele[cxc0+c]
            x++
            cxc0 += a.stride
        }
        // calcualate diagonal element
        sum := a.ele[x]
        for k := 0; k < r; k++ {
            sum -= l.ele[rxc0+k] * cmplx.Conj(l.ele[rxc0+k])
        }
        l.ele[x] = cmplx.Sqrt(sum)
        rxc0 += a.stride
    }
    return l
}

func main() {
    demo("A:", matrixFromRows([][]complex128{
        {25, 15, -5},
        {15, 18, 0},
        {-5, 0, 11},
    }))
    demo("A:", matrixFromRows([][]complex128{
        {18, 22, 54, 42},
        {22, 70, 86, 62},
        {54, 86, 174, 134},
        {42, 62, 134, 106},
    }))
}

func demo(heading string, a *matrix) {
    a.print(heading)
    a.choleskyDecomp().print("Cholesky factor L:")
}
```

Output:

```
A:
[(  25.00  +0.00i) ( +15.00  +0.00i) (  -5.00  +0.00i)]
[(  15.00  +0.00i) ( +18.00  +0.00i) (  +0.00  +0.00i)]
```

```
[(  -5.00  +0.00i) (  +0.00  +0.00i) ( +11.00  +0.00i)]

Cholesky factor L:
[(   5.00  +0.00i) (  +0.00  +0.00i) (  +0.00  +0.00i)]
[(   3.00  +0.00i) (  +3.00  +0.00i) (  +0.00  +0.00i)]
[(  -1.00  +0.00i) (  +1.00  +0.00i) (  +3.00  +0.00i)]

A:
[(  18.00  +0.00i) ( +22.00  +0.00i) ( +54.00  +0.00i) ( +42.00  +0.00i)]
[(  22.00  +0.00i) ( +70.00  +0.00i) ( +86.00  +0.00i) ( +62.00  +0.00i)]
[(  54.00  +0.00i) ( +86.00  +0.00i) (+174.00  +0.00i) (+134.00  +0.00i)]
[(  42.00  +0.00i) ( +62.00  +0.00i) (+134.00  +0.00i) (+106.00  +0.00i)]

Cholesky factor L:
[(   4.24  +0.00i) (  +0.00  +0.00i) (  +0.00  +0.00i) (  +0.00  +0.00i)]
[(   5.19  +0.00i) (  +6.57  +0.00i) (  +0.00  +0.00i) (  +0.00  +0.00i)]
[(  12.73  +0.00i) (  +3.05  +0.00i) (  +1.65  +0.00i) (  +0.00  +0.00i)]
[(   9.90  +0.00i) (  +1.62  +0.00i) (  +1.85  +0.00i) (  +1.39  +0.00i)]
```

Library

```go
package main

import (
    "fmt"

    mat "github.com/skelterjohn/go.matrix"
)

func main() {
    demo(mat.MakeDenseMatrix([]float64{
        25, 15, -5,
        15, 18, 0,
        -5, 0, 11,
    }, 3, 3))
    demo(mat.MakeDenseMatrix([]float64{
        18, 22, 54, 42,
        22, 70, 86, 62,
        54, 86, 174, 134,
        42, 62, 134, 106,
    }, 4, 4))
}

func demo(m *mat.DenseMatrix) {
    fmt.Println("A:")
    fmt.Println(m)
    l, err := m.Cholesky()
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("L:")
    fmt.Println(l)
}
```

Output:

```
A:
{25, 15, -5,
 15, 18,  0,
 -5,  0, 11}
L:
{ 5,  0,  0,
  3,  3,  0,
 -1,  1,  3}
A:
{ 18,  22,  54,  42,
  22,  70,  86,  62,
  54,  86, 174, 134,
  42,  62, 134, 106}
L:
{ 4.242641,         0,         0,         0,
   5.18545,  6.565905,         0,         0,
 12.727922,  3.046038,  1.649742,         0,
  9.899495,  1.624554,  1.849711,  1.392621}
```

Haskell

This example is **incomplete**. Conspicuous by being the only example that does not attempt to distinguish rows in matrix output. Please ensure that it meets all task requirements and remove this message.

We use the Cholesky–Banachiewicz algorithm (http://en.wikipedia.org/wiki/Cholesky_decomposition#The_Cholesky.E2.80.93Banachiewicz_and_Cholesky.E2.80.93Crout_algorithms) described in the Wikipedia article.

For more serious numerical analysis there is a Cholesky decomposition function in the hmatrix package (http://hackage.haskell.org/package/hmatrix) .

The Cholesky module:

```
module Cholesky (Arr, cholesky) where

import Data.Array.IArray
import Data.Array.MArray
import Data.Array.Unboxed
import Data.Array.ST

type Idx = (Int,Int)
type Arr = UArray Idx Double

-- Return the (i,j) element of the lower triangular matrix.  (We assume the
-- lower array bound is (0,0).)
get :: Arr -> Arr -> Idx -> Double
get a l (i,j) | i == j = sqrt $ a!(j,j) - dot
              | i  > j = (a!(i,j) - dot) / l!(j,j)
              | otherwise = 0
  where dot = sum [l!(i,k) * l!(j,k) | k <- [0..j-1]]

-- Return the lower triangular matrix of a Cholesky decomposition.  We assume
-- the input is a real, symmetric, positive-definite matrix, with lower array
-- bounds of (0,0).
cholesky :: Arr -> Arr
cholesky a = let n = maxBnd a
             in runSTUArray $ do
               l <- thaw a
               mapM_ (update a l) [(i,j) | i <- [0..n], j <- [0..n]]
               return l
  where maxBnd = fst . snd . bounds
        update a l i = unsafeFreeze l >>= \l' -> writeArray l i (get a l' i)
```

The main module:

```
import Data.Array.IArray
import Cholesky

ex1, ex2 :: Arr
ex1 = listArray ((0,0),(2,2)) [25, 15, -5,
                               15, 18,  0,
                               -5,  0, 11]

ex2 = listArray ((0,0),(3,3)) [18, 22,  54,  42,
                               22, 70,  86,  62,
                               54, 86, 174, 134,
                               42, 62, 134, 106]

main :: IO ()
main = do
  print $ elems $ cholesky ex1
  print $ elems $ cholesky ex2
```

The resulting matrices are printed as lists, as in the following output:

```
[5.0,0.0,0.0,3.0,3.0,0.0,-1.0,1.0,3.0]
[4.242640687119285,0.0,0.0,0.0,5.185449728701349,6.565905201197403,0.0,0.0,12.727922061357857,3.0460384954008553,1.6497422479090704,0.0,9
```

Icon and Unicon

```
procedure cholesky (array)
  result := make_square_array (*array)
  every (i := 1 to *array) do {
    every (k := 1 to i) do {
      sum := 0
```

```
      every (j := 1 to (k-1)) do {
        sum +:= result[i][j] * result[k][j]
      }
      if (i = k)
        then result[i][k] := sqrt(array[i][i] - sum)
        else result[i][k] := 1.0 / result[k][k] * (array[i][k] - sum)
    }
  }
  return result
end

procedure make_square_array (n)
  result := []
  every (1 to n) do push (result, list(n, 0))
  return result
end

procedure print_array (array)
  every (row := !array) do {
    every writes (!row || " ")
    write ()
  }
end

procedure do_cholesky (array)
  write ("Input:")
  print_array (array)
  result := cholesky (array)
  write ("Result:")
  print_array (result)
end

procedure main ()
  do_cholesky ([[25,15,-5],[15,18,0],[-5,0,11]])
  do_cholesky ([[18,22,54,42],[22,70,86,62],[54,86,174,134],[42,62,134,106]])
end
```

Output:

```
Input:
25 15 -5
15 18 0
-5 0 11
Result:
5.0 0 0
3.0 3.0 0
-1.0 1.0 3.0
Input:
18 22 54 42
22 70 86 62
54 86 174 134
42 62 134 106
Result:
4.242640687 0 0 0
5.185449729 6.565905201 0 0
12.72792206 3.046038495 1.649742248 0
9.899494937 1.624553864 1.849711005 1.392621248
```

J

Solution:

```
mp=: +/ . *  NB. matrix product
h =: +@|:    NB. conjugate transpose

cholesky=: 3 : 0
 n=. #A=. y
 if. 1>:n do.
  assert. (A=|A)>0=A  NB. check for positive definite
  %:A
 else.
  'X Y t Z'=. , (;~n$(>.-:n){.1) <;.1 A
  L0=. cholesky X
  L1=. cholesky Z-(T=.(h Y) mp %.X) mp Y
  L0,(T mp L0),.L1
 end.
)
```

See Cholesky Decomposition essay on the J Wiki.

Examples:

```
   eg1=: 25 15 _5 , 15 18 0 ,: _5 0 11
   eg2=: 18 22 54 42 , 22 70 86 62 , 54 86 174 134 ,: 42 62 134 106
   cholesky eg1
 5 0 0
 3 3 0
_1 1 3
   cholesky eg2
4.24264       0       0       0
5.18545 6.56591       0       0
12.7279 3.04604 1.64974       0
9.89949 1.62455 1.84971 1.39262
```

Java

Works with: Java version 1.5+

```
import java.util.Arrays;

public class Cholesky {
	public static double[][] chol(double[][] a){
		int m = a.length;
		double[][] l = new double[m][m]; //automatically initialzed to 0's
		for(int i = 0; i< m;i++){
			for(int k = 0; k < (i+1); k++){
				double sum = 0;
				for(int j = 0; j < k; j++){
					sum += l[i][j] * l[k][j];
				}
				l[i][k] = (i == k) ? Math.sqrt(a[i][i] - sum) :
					(1.0 / l[k][k] * (a[i][k] - sum));
			}
		}
		return l;
	}

	public static void main(String[] args){
		double[][] test1 = {{25, 15, -5},
		                                      {15, 18, 0},
		                                      {-5, 0, 11}};
		System.out.println(Arrays.deepToString(chol(test1)));
		double[][] test2 = {{18, 22, 54, 42},
		                                      {22, 70, 86, 62},
		                                      {54, 86, 174, 134},
		                                      {42, 62, 134, 106}};
		System.out.println(Arrays.deepToString(chol(test2)));
	}
}
```

Output:

```
[[5.0, 0.0, 0.0], [3.0, 3.0, 0.0], [-1.0, 1.0, 3.0]]
[[4.242640687119285, 0.0, 0.0, 0.0], [5.185449728701349, 6.565905201197403, 0.0, 0.0], [12.727922061357857, 3.0460384954008553, 1.6497422
```

jq

Works with: jq version 1.4

Infrastructure:

```
# Create an m x n matrix
 def matrix(m; n; init):
   if m == 0 then []
   elif m == 1 then [range(0; n)] | map(init)
   elif m > 0 then
     matrix(1; n; init) as $row
     | [range(0; m)] | map( $row )
   else error("matrix\(m);_;_) invalid")
   end ;

# Print a matrix neatly, each cell ideally occupying n spaces,
# but without truncation
def neatly(n):
  def right: tostring | ( " " * (n-length) + .);
```

```
  . as $in
  | length as $length
  | reduce range (0; $length) as $i
      (""; . + reduce range(0; $length) as $j
      (""; "\(.) \($in[$i][$j] | right )" ) + "\n" ) ;

def is_square:
  type == "array" and (map(type == "array") | all) and
    length == 0 or ( (.[0]|length) as $l | map(length == $l) | all) ;

# This implementation of is_symmetric/0 uses a helper function that circumvents
# limitations of jq 1.4:
def is_symmetric:
    # [matrix, i,j, len]
    def test:
       if .[1] > .[3] then true
       elif .[1] == .[2] then [ .[0], .[1] + 1, 0, .[3]] | test
       elif .[0][.[1]][.[2]] == .[0][.[2]][.[1]]
         then [ .[0], .[1], .[2]+1, .[3]] | test
      else false
      end;
    if is_square|not then false
    else [ ., 0, 0, length ] | test
    end ;
```

Cholesky Decomposition:

```
def cholesky_factor:
  if is_symmetric then
    length as $length
    | . as $self
    | reduce range(0; $length) as $k
        ( matrix(length; length; 0); # the matrix that will hold the answer
          reduce range(0; $length) as $i
            (.;
             if $i == $k
               then (. as $lower
                     | reduce range(0; $k) as $j
                         (0; . + ($lower[$k][$j] | .*.) )) as $sum
                 | .[$k][$k] = (($self[$k][$k] - $sum) | sqrt)
             elif $i > $k
               then (. as $lower
                     | reduce range(0; $k) as $j
                         (0; . + $lower[$i][$j] * $lower[$k][$j])) as $sum
                 | .[$i][$k] = (($self[$k][$i] - $sum) / .[$k][$k] )
             else .
             end ))
  else error( "cholesky_factor: matrix is not symmetric" )
  end ;
```

Task 1:

```
[[25,15,-5],[15,18,0],[-5,0,11]] | cholesky_factor
```

Output:

```
[[5,0,0],[3,3,0],[-1,1,3]]
```

Task 2:

```
[[18, 22,  54,  42],
 [22, 70,  86,  62],
 [54, 86, 174, 134],
 [42, 62, 134, 106]] | cholesky_factor | neatly(20)
```

Output:

```
    4.242640687119285                    0                    0                    0
    5.185449728701349    6.565905201197403                    0                    0
   12.727922061357857   3.0460384954008553   1.6497422479090704                    0
    9.899494936611665   1.6245538642137891    1.849711005231382   1.3926212476455924
```

Julia

Julia's strong linear algebra support includes Cholesky decomposition.

```
a = [25 15 5; 15 18 0; -5 0 11]
b = [18 22 54 22; 22 70 86 62; 54 86 174 134; 42 62 134 106]

println(a, "\n => \n", chol(a, :L))
println(b, "\n => \n", chol(b, :L))
```

Output:

```
[25 15 5
 15 18 0
 -5 0 11]
 =>
[5.0 0.0 0.0
 3.0 3.0 0.0
 -1.0 1.0 3.0]
[18 22 54 22
 22 70 86 62
 54 86 174 134
 42 62 134 106]
 =>
[4.242640687119285 0.0 0.0 0.0
 5.185449728701349 6.565905201197403 0.0 0.0
 12.727922061357857 3.0460384954008553 1.6497422479090704 0.0
 9.899494936611667 1.624553864213788 1.8497110052313648 1.3926212476456026]
```

Maple

The Cholesky decomposition is obtained by passing the `method = Cholesky' option to the LUDecomposition procedure in the LinearAlgebra pacakge. This is illustrated below for the two requested examples. The first is computed exactly; the second is also, but the subsequent application of `evalf' to the result produces a matrix with floating point entries which can be compared with the expected output in the problem statement.

```
> A := << 25, 15, -5; 15, 18, 0; -5, 0, 11 >>;
                              [25    15    -5]
                              [              ]
                         A := [15    18     0]
                              [              ]
                              [-5     0    11]

> B := << 18, 22, 54, 42; 22, 70, 86, 62; 54, 86, 174, 134; 42, 62, 134, 106>>;
                          [18    22     54     42]
                          [                      ]
                          [22    70     86     62]
                     B := [                      ]
                          [54    86    174    134]
                          [                      ]
                          [42    62    134    106]

> use LinearAlgebra in
>       LUDecomposition( A, method = Cholesky );
>       LUDecomposition( B, method = Cholesky );
>       evalf( % );
> end use;
                             [ 5    0    0]
                             [            ]
                             [ 3    3    0]
                             [            ]
                             [-1    1    3]

             [   1/2                                 ]
             [3 2          0           0           0   ]
             [                                       ]
             [    1/2        1/2                     ]
             [11 2       2 97                        ]
             [-------    -------         0           0   ]
             [   3          3                        ]
             [                                       ]
             [                1/2         1/2        ]
             [   1/2     30 97       2 6402            ]
             [9 2        --------    ---------       0   ]
             [              97         97            ]
             [                                       ]
             [                1/2          1/2      1/2]
             [   1/2     16 97       74 6402      8 33   ]
             [7 2        --------    ----------    -------]
```

```
            [               97          3201         33   ]

      [4.242640686        0.             0.             0.     ]
      [                                                        ]
      [5.185449728    6.565905202        0.             0.     ]
      [                                                        ]
      [12.72792206    3.046038495    1.649742248        0.     ]
      [                                                        ]
      [9.899494934    1.624553864    1.849711006    1.392621248]
```

Mathematica

```
CholeskyDecomposition[{{25, 15, -5}, {15, 18, 0}, {-5, 0, 11}}]
```

MATLAB / Octave

The cholesky decomposition chol() is an internal function

```
  A = [
  25  15  -5
  15  18   0
  -5   0  11 ];

  B  = [
  18  22   54   42
  22  70   86   62
  54  86  174  134
  42  62  134  106   ];

  [L] = chol(A,'lower')
  [L] = chol(B,'lower')
```

Output:

```
   >   [L] = chol(A,'lower')
  L =

   5   0   0
   3   3   0
  -1   1   3

  >   [L] = chol(B,'lower')
  L =
    4.24264    0.00000    0.00000    0.00000
    5.18545    6.56591    0.00000    0.00000
   12.72792    3.04604    1.64974    0.00000
    9.89949    1.62455    1.84971    1.39262
```

Maxima

```
/* Cholesky decomposition is built-in */

a: hilbert_matrix(4)$

b: cholesky(a);
/* matrix([1,   0,              0,             0             ],
          [1/2, 1/(2*sqrt(3)), 0,             0             ],
          [1/3, 1/(2*sqrt(3)), 1/(6*sqrt(5)), 0             ],
          [1/4, 3^(3/2)/20,    1/(4*sqrt(5)), 1/(20*sqrt(7))]) */

b . transpose(b) - a;
matrix([0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0])
```

Nim

Translation of: C

```
import math, strutils

proc cholesky[T](a: T): T =
  for i in 0 .. < a[0].len:
    for j in 0 .. i:
      var s = 0.0
      for k in 0 .. < j:
        s += result[i][k] * result[j][k]
      result[i][j] = if i == j: sqrt(a[i][i]-s)
        else: (1.0 / result[j][j] * (a[i][j] - s))

proc `$`(a): string =
  result = ""
  for b in a:
    for c in b:
      result.add c.formatFloat(ffDecimal, 5) & " "
    result.add "\n"

let m1 = [[25.0, 15.0, -5.0],
          [15.0, 18.0,  0.0],
          [-5.0,  0.0, 11.0]]
echo cholesky(m1)

let m2 = [[18.0, 22.0,  54.0,  42.0],
          [22.0, 70.0,  86.0,  62.0],
          [54.0, 86.0, 174.0, 134.0],
          [42.0, 62.0, 134.0, 106.0]]
echo cholesky(m2)
```

Output:

```
5.00000 0.00000 0.00000
3.00000 3.00000 0.00000
-1.00000 1.00000 3.00000

4.24264 0.00000 0.00000 0.00000
5.18545 6.56591 0.00000 0.00000
12.72792 3.04604 1.64974 0.00000
9.89949 1.62455 1.84971 1.39262
```

Objeck

Translation of: C

```
class Cholesky {
  function : Main(args : String[]) ~ Nil {
    n := 3;
    m1 := [25.0, 15.0, -5.0, 15.0, 18.0, 0.0, -5.0, 0.0, 11.0];
    c1 := Cholesky(m1, n);
    ShowMatrix(c1, n);

    IO.Console->PrintLine();

    n := 4;
    m2 := [18.0, 22.0,  54.0,  42.0, 22.0, 70.0, 86.0, 62.0,
      54.0, 86.0, 174.0, 134.0, 42.0, 62.0, 134.0, 106.0];
    c2 := Cholesky(m2, n);
    ShowMatrix(c2, n);
  }

  function : ShowMatrix(A : Float[], n : Int) ~ Nil {
    for (i := 0; i < n; i+=1;) {
      for (j := 0; j < n; j+=1;) {
        IO.Console->Print(A[i * n + j])->Print('\t');
      };
      IO.Console->PrintLine();
    };
  }

  function : Cholesky(A : Float[], n : Int) ~ Float[] {
    L := Float->New[n * n];

    for (i := 0; i < n; i+=1;) {
      for (j := 0; j < (i+1); j+=1;) {
        s := 0.0;
        for (k := 0; k < j; k+=1;) {
          s += L[i * n + k] * L[j * n + k];
        };
        L[i * n + j] := (i = j) ?
          (A[i * n + i] - s)->SquareRoot() :
          (1.0 / L[j * n + j] * (A[i * n + j] - s));
      };
```

```
    };

    return L;
  }
}
```

```
5       0       0
3       3       0
-1      1       3

4.24264069      0               0               0
5.18544973      6.5659052       0               0
12.7279221      3.0460385       1.64974225      0
9.89949494      1.62455386      1.84971101      1.39262125
```

OCaml

```
let cholesky inp =
   let n = Array.length inp in
   let res = Array.make_matrix n n 0.0 in
   let factor i k =
      let rec sum j =
         if j = k then 0.0 else
         res.(i).(j) *. res.(k).(j) +. sum (j+1) in
      inp.(i).(k) -. sum 0 in
   for col = 0 to n-1 do
      res.(col).(col) <- sqrt (factor col col);
      for row = col+1 to n-1 do
         res.(row).(col) <- (factor row col) /. res.(col).(col)
      done
   done;
   res

let pr_vec v = Array.iter (Printf.printf " %9.5f") v; print_newline()
let show = Array.iter pr_vec
let test a =
   print_endline "\nin:"; show a;
   print_endline "out:"; show (cholesky a)

let _ =
   test [| [|25.0; 15.0; -5.0|];
           [|15.0; 18.0;  0.0|];
           [|-5.0;  0.0; 11.0|] |];
   test [| [|18.0; 22.0;  54.0;  42.0|];
           [|22.0; 70.0;  86.0;  62.0|];
           [|54.0; 86.0; 174.0; 134.0|];
           [|42.0; 62.0; 134.0; 106.0|] |];
```

Output:

```
in:
  25.00000  15.00000  -5.00000
  15.00000  18.00000   0.00000
  -5.00000   0.00000  11.00000
out:
   5.00000   0.00000   0.00000
   3.00000   3.00000   0.00000
  -1.00000   1.00000   3.00000

in:
  18.00000  22.00000  54.00000  42.00000
  22.00000  70.00000  86.00000  62.00000
  54.00000  86.00000 174.00000 134.00000
  42.00000  62.00000 134.00000 106.00000
out:
   4.24264   0.00000   0.00000   0.00000
   5.18545   6.56591   0.00000   0.00000
  12.72792   3.04604   1.64974   0.00000
   9.89949   1.62455   1.84971   1.39262
```

Pascal

```
Program Cholesky;

type
  D2Array = array of array of double;
```

```
function cholesky(const A: D2Array): D2Array;
  var
    i, j, k: integer;
    s: double;
  begin
    setlength(cholesky, length(A), length(A));
    for i := low(cholesky) to high(cholesky) do
      for j := 0 to i do
      begin
        s := 0;
        for k := 0 to j - 1 do
          s := s + cholesky[i][k] * cholesky[j][k];
        if i = j then
          cholesky[i][j] := sqrt(A[i][i] - s)
        else
          cholesky[i][j] := (A[i][j] - s) / cholesky[j][j];  // save one multiplication compared to the original
      end;
  end;

procedure printM(const A: D2Array);
  var
    i, j: integer;
  begin
    for i :=  low(A) to high(A) do
    begin
      for j := low(A) to high(A) do
        write(A[i,j]:8:5);
      writeln;
    end;
  end;

const
  m1: array[0..2,0..2] of double = ((25, 15, -5),
                                    (15, 18,  0),
                                    (-5,  0, 11));
  m2: array[0..3,0..3] of double = ((18, 22,  54,  42),
                                    (22, 70,  86,  62),
                                    (54, 86, 174, 134),
                                    (42, 62, 134, 106));
var
  index: integer;
  cIn, cOut: D2Array;

begin
  setlength(cIn, length(m1), length(m1));
  for index := low(m1) to high(m1) do
    cIn[index] := m1[index];
  cOut := cholesky(cIn);
  printM(cOut);

  writeln;

  setlength(cIn, length(m2), length(m2));
  for index := low(m2) to high(m2) do
    cIn[index] := m2[index];
  cOut := cholesky(cIn);
  printM(cOut);

end.
```

Output:

```
 5.00000 0.00000 0.00000
 3.00000 3.00000 0.00000
-1.00000 1.00000 3.00000

 4.24264 0.00000 0.00000 0.00000
 5.18545 6.56591 0.00000 0.00000
12.72792 3.04604 1.64974 0.00000
 9.89949 1.62455 1.84971 1.39262
```

Perl

```
sub cholesky {
  my $matrix = shift;
  my $chol = [ map { [(0) x @$matrix ] } @$matrix ];
  for my $row (0..@$matrix-1) {
    for my $col (0..$row) {
      my $x = $$matrix[$row][$col];
      $x -= $$chol[$row][$_]*$$chol[$col][$_] for 0..$col;
      $$chol[$row][$col] = $row == $col ? sqrt $x : $x/$$chol[$col][$col];
    }
```

```
  }
  return $chol;
}

my $example1 = [ [ 25, 15, -5 ],
                 [ 15, 18,  0 ],
                 [ -5,  0, 11 ] ];
print "Example 1:\n";
print +(map { sprintf "%7.4f\t", $_ } @$_), "\n" for @{ cholesky $example1 };

my $example2 = [ [ 18, 22,  54,  42],
                 [ 22, 70,  86,  62],
                 [ 54, 86, 174, 134],
                 [ 42, 62, 134, 106] ];
print "\nExample 2:\n";
print +(map { sprintf "%7.4f\t", $_ } @$_), "\n" for @{ cholesky $example2 };
```

Output:

```
Example 1:
 5.0000  0.0000  0.0000
 3.0000  3.0000  0.0000
-1.0000  1.0000  3.0000

Example 2:
 4.2426  0.0000  0.0000  0.0000
 5.1854  6.5659  0.0000  0.0000
12.7279  3.0460  1.6497  0.0000
 9.8995  1.6246  1.8497  1.3926
```

Perl 6

```
sub cholesky(@A) {
    my @L = @A »*» 0;
    for ^@A -> $i {
        for 0..$i -> $j {
            @L[$i][$j] = ($i == $j ?? &sqrt !! 1/@L[$j][$j] * * )(
                @A[$i][$j] - [+] (@L[$i;*] Z* @L[$j;*])[^$j]
            );
        }
    }
    return @L;
}
.say for cholesky [
    [25],
    [15, 18],
    [-5,  0, 11],
];

.say for cholesky [
    [18, 22,  54,  42],
    [22, 70,  86,  62],
    [54, 86, 174, 134],
    [42, 62, 134, 106],
];
```

PicoLisp

```
(scl 9)
(load "@lib/math.l")

(de cholesky (A)
   (let L (mapcar '(() (need (length A) 0)) A)
      (for (I . R) A
         (for J I
            (let S (get R J)
               (for K (inc J)
                  (dec 'S (*/ (get L I K) (get L J K) 1.0)) )
               (set (nth L I J)
                  (if (= I J)
                     (sqrt S 1.0)
                     (*/ S 1.0 (get L J J)) ) ) ) ) )
      (for R L
         (for N R (prin (align 9 (round N 5))))
         (prinl) ) ) )
```

Test:

```
(cholesky
   '((25.0 15.0 -5.0) (15.0 18.0 0) (-5.0 0 11.0)) )

(prinl)

(cholesky
   (quote
      (18.0  22.0   54.0   42.0)
      (22.0  70.0   86.0   62.0)
      (54.0  86.0  174.0  134.0)
      (42.0  62.0  134.0  106.0) ) )
```

Output:

```
  5.00000  0.00000  0.00000
  3.00000  3.00000  0.00000
 -1.00000  1.00000  3.00000

  4.24264  0.00000  0.00000  0.00000
  5.18545  6.56591  0.00000  0.00000
 12.72792  3.04604  1.64974  0.00000
  9.89949  1.62455  1.84971  1.39262
```

PL/I

```
(subscriptrange):
decompose: procedure options (main);   /* 31 October 2013 */
   declare a(*,*) float controlled;

   allocate a(3,3) initial (25, 15, -5,
                            15, 18,  0,
                            -5,  0, 11);
    put skip list ('Original matrix:');
    put edit (a) (skip, 3 f(4));

    call cholesky(a);
    put skip list ('Decomposed matrix');
    put edit (a) (skip, 3 f(4));
    free a;
    allocate a(4,4) initial (18, 22,  54,  42,
                             22, 70,  86,  62,
                             54, 86, 174, 134,
                             42, 62, 134, 106);
    put skip list ('Original matrix:');
    put edit (a) (skip, (hbound(a,1)) f(12) );
    call cholesky(a);
    put skip list ('Decomposed matrix');
    put edit (a) (skip, (hbound(a,1)) f(12,5) );

cholesky: procedure(a);
   declare a(*,*) float;
   declare L(hbound(a,1), hbound(a,2)) float;
   declare s float;
   declare (i, j, k) fixed binary;

   L = 0;
   do i = lbound(a,1) to hbound(a,1);
      do j = lbound(a,2) to i;
         s = 0;
         do k = lbound(a,2) to j-1;
            s = s + L(i,k) * L(j,k);
         end;
         if i = j then
            L(i,j) = sqrt(a(i,i) - s);
         else
            L(i,j) = (a(i,j) - s) / L(j,j);
      end;
   end;
   a = L;
end cholesky;

end decompose;
```

ACTUAL RESULTS:-

```
Original matrix:
  25  15  -5
```

```
 15  18   0
 -5   0  11
Decomposed matrix
  5   0   0
  3   3   0
 -1   1   3
Original matrix:
          18          22          54          42
          22          70          86          62
          54          86         174         134
          42          62         134         106
Decomposed matrix
     4.24264     0.00000     0.00000     0.00000
     5.18545     6.56591     0.00000     0.00000
    12.72792     3.04604     1.64974     0.00000
     9.89950     1.62455     1.84971     1.39262
```

Python

Python2.X version

```python
from __future__ import print_function

from pprint import pprint
from math import sqrt


def cholesky(A):
    L = [[0.0] * len(A) for _ in xrange(len(A))]
    for i in xrange(len(A)):
        for j in xrange(i+1):
            s = sum(L[i][k] * L[j][k] for k in xrange(j))
            L[i][j] = sqrt(A[i][i] - s) if (i == j) else \
                      (1.0 / L[j][j] * (A[i][j] - s))
    return L

if __name__ == "__main__":
    m1 = [[25, 15, -5],
          [15, 18,  0],
          [-5,  0, 11]]
    pprint(cholesky(m1))
    print()

    m2 = [[18, 22,  54,  42],
          [22, 70,  86,  62],
          [54, 86, 174, 134],
          [42, 62, 134, 106]]
    pprint(cholesky(m2), width=120)
```

Output:

```
[[5.0, 0.0, 0.0], [3.0, 3.0, 0.0], [-1.0, 1.0, 3.0]]

[[4.242640687119285, 0.0, 0.0, 0.0],
 [5.185449728701349, 6.565905201197403, 0.0, 0.0],
 [12.727922061357857, 3.0460384954008553, 1.6497422479090704, 0.0],
 [9.899494936611667, 1.624553864213788, 1.8497110052313648, 1.3926212476456026]]
```

Python3.X version using extra Python idioms

Factors out accesses to `A[i], L[i], and L[j]` by creating `Ai, Li and Lj` respectively as well as using `enumerate` instead of `range(len(some_array))`.

```python
def cholesky(A):
    L = [[0.0] * len(A) for _ in range(len(A))]
    for i, (Ai, Li) in enumerate(zip(A, L)):
        for j, Lj in enumerate(L[:i+1]):
            s = sum(Li[k] * Lj[k] for k in range(j))
            Li[j] = sqrt(Ai[i] - s) if (i == j) else \
                      (1.0 / Lj[j] * (Ai[j] - s))
    return L
```

Output:

(As above)

q

```
solve:{[A;B] $[0h>type A;B%A;inv[A] mmu B]}
ak:{[m;k] (),/:m[;k]til k:k-1}
akk:{[m;k] m[k;k:k-1]}
transpose:{$[0h=type x;flip x;enlist each x]}
mult:{[A;B]$[0h=type A;A mmu B;A*B]}
cholesky:{[A]
        {[A;L;n]
                l_k:solve[L;ak[A;n]];
                l_kk:first over sqrt[akk[A;n] - mult[transpose l_k;l_k]];
                ({$[0h<type x;enlist x;x]}L,'0f),enlist raze transpose[l_k],l_kk
                }[A]/[sqrt A[0;0];1_1+til count first A]
        }

show cholesky (25 15 -5f;15 18 0f;-5 0 11f)
-1"";
show cholesky (18 22 54 42f;22 70 86 62f;54 86 174 134f;42 62 134 106f)
```

Output:

```
5  0 0
3  3 0
-1 1 3

4.242641 0        0        0
5.18545  6.565905 0        0
12.72792 3.046038 1.649742 0
9.899495 1.624554 1.849711 1.392621
```

R

```
t(chol(matrix(c(25, 15, -5, 15, 18, 0, -5, 0, 11), nrow=3, ncol=3)))
#      [,1] [,2] [,3]
# [1,]    5    0    0
# [2,]    3    3    0
# [3,]   -1    1    3

t(chol(matrix(c(18, 22, 54, 42, 22, 70, 86, 62, 54, 86, 174, 134, 42, 62, 134, 106), nrow=4, ncol=4)))
#           [,1]     [,2]     [,3]     [,4]
# [1,]  4.242641 0.000000 0.000000 0.000000
# [2,]  5.185450 6.565905 0.000000 0.000000
# [3,] 12.727922 3.046038 1.649742 0.000000
# [4,]  9.899495 1.624554 1.849711 1.392621
```

Racket

```
#lang racket
(require math)

(define (cholesky A)
  (define mref matrix-ref)
  (define n (matrix-num-rows A))
  (define L (for/vector ([_ n]) (for/vector ([_ n]) 0)))
  (define (set L i j x) (vector-set! (vector-ref L i) j x))
  (define (ref L i j) (vector-ref (vector-ref L i) j))
  (for* ([i n] [k n])
    (set L i k
         (cond
           [(= i k)
            (sqrt (- (mref A i i) (for/sum ([j k]) (sqr (ref L k j)))))]
           [(> i k)
            (/ (- (mref A i k) (for/sum ([j k]) (* (ref L i j) (ref L k j))))
               (ref L k k))]
           [else 0])))
  L)

(cholesky (matrix [[25 15 -5]
                   [15 18  0]
                   [-5  0 11]]))

(cholesky (matrix [[18 22  54 42]
                   [22 70  86 62]
                   [54 86 174 134]
                   [42 62 134 106]]))
```

Output:

```
'#(#(5 0 0)
  #(3 3 0)
  #(-1 1 3))
'#(#(4.242640687119285 0 0 0)
  #( 5.185449728701349 6.565905201197403  0                  0)
  #(12.727922061357857 3.0460384954008553 1.6497422479090704 0)
  #( 9.899494936611665 1.6245538642137891 1.849711005231382  1.3926212476455924))
```

REXX

If trailing zeroes are wanted after the decimal point for values of zero (0), the **/1** (division by one performs REXX number normalization) can be removed from the line (number 73) which contains the source statement:

```
aLine=aLine right( format(@.row.col,, decPlaces) /1, width)
```

```
/*REXX program to perform the  Cholesky  decomposition on square matrix.*/

niner= '25  15  -5' ,
       '15  18   0' ,
       '-5   0  11'
                            call Cholesky niner
hexer= 18  22  54  42,
       22  70  86  62,
       54  86 174 134,
       42  62 134 106
                            call Cholesky hexer
exit                                   /*stick a fork in it, we're done.*/
/*─────────────────────────────────────Cholesky subroutine──────────────*/
Cholesky: procedure; arg !; call tell 'input array',!

         do row=1 for order
                do col=1 for row; s=0
                            do i=1 for col-1
                            s=s+$.row.i*$.col.i
                            end   /*i*/
                if row=col then $.row.row=sqrt($.row.row-s)
                           else $.row.col=1/$.col.col*(@.row.col-s)
                end   /*col*/
         end          /*row*/

call tell 'Cholesky factor',,$.,'─'
return
/*─────────────────────────────────────TELL subroutine───&find the order*/
tell:  parse arg hdr,x,y,sep;   #=0;   if sep=='' then sep='='
decPlaces =  5       /*number of decimal places past the decimal point. */
width     = 10       /*width of field to be used to display the elements*/

if y=='' then    $.=0
         else    do row=1      for order
                       do col=1 for order
                       x=x  $.row.col
                       end   /*row*/
                 end         /*col*/
w=words(x)

     do order=1 until order**2>=w      /*fast way to find the MAT order.*/
     end   /*order*/

if order**2\==w then call err "matrix elements don't match its order"
say;    say center(hdr, ((width+1)*w)%order, sep);    say

        do row=1       for order;     aLine=
             do col=1 for order;     #=#+1
                                  @.row.col=word(x,#)
             if col<=row  then  $.row.col=@.row.col
             aLine=aLine  right( format(@.row.col,, decPlaces) /1, width)
             end   /*col*/
        say aLine
        end        /*row*/
return
/*─────────────────────────────────────SQRT subroutine──────────────────*/
sqrt: procedure;   parse arg x;   if x=0 then return 0;   d=digits()
  numeric digits 11; g=.sqrtGuess(); do j=0 while p>9; m.j=p; p=p%2+1; end
    do k=j+5 to 0 by -1;if m.k>11 then numeric digits m.k;g=.5*(g+x/g);end
  numeric digits d;    return g/1
```

```
.sqrtGuess: if x<0  then call err 'SQRT of negative #';    numeric form
        m.=11; p=d+d%4+2; parse value format(x,2,1,,0) 'E0' with g 'E' _ .
        return g*.5'E'_%2
/*────────────────────────────────────────ERR subroutine───────────────────*/
err:  say; say;  say '***error***!';  say;  say arg(1);  say; say; exit 13
```

Output:

```
═══════════input array═══════════

         25         15         -5
         15         18          0
         -5          0         11

─────────Cholesky factor─────────

          5          0          0
          3          3          0
         -1          1          3

════════════════input array════════════════

         18         22         54         42
         22         70         86         62
         54         86        174        134
         42         62        134        106

──────────────Cholesky factor───────────────

    4.24264          0          0          0
    5.18545    6.56591          0          0
   12.72792    3.04604    1.64974          0
    9.89949    1.62455    1.84971    1.39262
```

Ruby

```ruby
require 'matrix'

class Matrix
  def symmetric?
    return false if not square?
    (0 ... row_size).each do |i|
      (0 .. i).each do |j|
        return false if self[i,j] != self[j,i]
      end
    end
    true
  end

  def cholesky_factor
    raise ArgumentError, "must provide symmetric matrix" unless symmetric?
    l = Array.new(row_size) {Array.new(row_size, 0)}
    (0 ... row_size).each do |k|
      (0 ... row_size).each do |i|
        if i == k
          sum = (0 .. k-1).inject(0.0) {|sum, j| sum + l[k][j] ** 2}
          val = Math.sqrt(self[k,k] - sum)
          l[k][k] = val
        elsif i > k
          sum = (0 .. k-1).inject(0.0) {|sum, j| sum + l[i][j] * l[k][j]}
          val = (self[k,i] - sum) / l[k][k]
          l[i][k] = val
        end
      end
    end
    Matrix[*l]
  end
end

puts Matrix[[25,15,-5],[15,18,0],[-5,0,11]].cholesky_factor
puts Matrix[[18, 22,  54,  42],
            [22, 70,  86,  62],
            [54, 86, 174, 134],
            [42, 62, 134, 106]].cholesky_factor
```

Output:

```
Matrix[[5.0, 0, 0], [3.0, 3.0, 0], [-1.0, 1.0, 3.0]]
Matrix[[4.242640687119285, 0, 0, 0],
```

```
 [5.185449728701349, 6.565905201197403, 0, 0],
 [12.727922061357857, 3.0460384954008553, 1.6497422479090704, 0],
 [9.899494936611665, 1.6245538642137891, 1.849711005231382, 1.3926212476455924]]
```

Scala

```scala
case class Matrix( val matrix:Array[Array[Double]] ) {

  // Assuming matrix is positive-definite, symmetric and not empty...

  val rows,cols = matrix.size

  def getOption( r:Int, c:Int ) : Option[Double] = Pair(r,c) match {
    case (r,c) if r < rows && c < rows => Some(matrix(r)(c))
    case _ => None
  }

  def isLowerTriangle( r:Int, c:Int ) : Boolean = { c <= r }
  def isDiagonal( r:Int, c:Int ) : Boolean = { r == c}

  override def toString = matrix.map(_.mkString(", ")).mkString("\n")

  /**
   * Perform Cholesky Decomposition of this matrix
   */
  lazy val cholesky : Matrix = {

    val l = Array.ofDim[Double](rows*cols)

    for( i <- (0 until rows); j <- (0 until cols) ) yield {

      val s = (for( k <- (0 until j) ) yield { l(i*rows+k) * l(j*rows+k) }).sum

      l(i*rows+j) = (i,j) match {
        case (r,c) if isDiagonal(r,c) => scala.math.sqrt(matrix(i)(i) - s)
        case (r,c) if isLowerTriangle(r,c) => (1.0 / l(j*rows+j) * (matrix(i)(j) - s))
        case _ => 0
      }
    }

    val m = Array.ofDim[Double](rows,cols)
    for( i <- (0 until rows); j <- (0 until cols) ) m(i)(j) = l(i*rows+j)
    Matrix(m)
  }
}

// A little test...
val a1 = Matrix(Array[Array[Double]](Array(25,15,-5),Array(15,18,0),Array(-5,0,11)))
val a2 = Matrix(Array[Array[Double]](Array(18,22,54,42), Array(22,70,86,62), Array(54,86,174,134), Array(42,62,134,106)))

val l1 = a1.cholesky
val l2 = a2.cholesky


// Given test results
val r1 = Array[Double](5,0,0,3,3,0,-1,1,3)
val r2 = Array[Double](4.24264,0.00000,0.00000,0.00000,5.18545,6.56591,0.00000,0.00000,
                        12.72792,3.04604,1.64974,0.00000,9.89949,1.62455,1.84971,1.39262)

// Verify assertions
(l1.matrix.flatten.zip(r1)).foreach{ case (result,test) =>
  assert(math.round( result * 100000 ) * 0.00001 == math.round( test * 100000 ) * 0.00001)
}

(l2.matrix.flatten.zip(r2)).foreach{ case (result,test) =>
  assert(math.round( result * 100000 ) * 0.00001 == math.round( test * 100000 ) * 0.00001)
}
```

Seed7

```
$ include "seed7_05.s7i";
  include "float.s7i";
  include "math.s7i";

const type: matrix is array array float;

const func matrix: cholesky (in matrix: a) is func
  result
    var matrix: cholesky is 0 times 0 times 0.0;
  local
    var integer: i is 0;
    var integer: j is 0;
```

```
    var integer: k is 0;
    var float: sum is 0.0;
  begin
    cholesky := length(a) times length(a) times 0.0;
    for key i range cholesky do
      for j range 1 to i do
        sum := 0.0;
        for k range 1 to j do
          sum +:= cholesky[i][k] * cholesky[j][k];
        end for;
        if i = j then
          cholesky[i][i] := sqrt(a[i][i] - sum)
        else
          cholesky[i][j] := (a[i][j] - sum) / cholesky[j][j];
        end if;
      end for;
    end for;
  end func;

const proc: writeMat (in matrix: a) is func
  local
    var integer: i is 0;
    var float: num is 0.0;
  begin
    for key i range a do
      for num range a[i] do
        write(num digits 5 lpad 8);
      end for;
      writeln;
    end for;
  end func;

const matrix: m1 is [] (
    [] (25.0, 15.0, -5.0),
    [] (15.0, 18.0,  0.0),
    [] (-5.0,  0.0, 11.0));
const matrix: m2 is [] (
    [] (18.0, 22.0,  54.0,  42.0),
    [] (22.0, 70.0,  86.0,  62.0),
    [] (54.0, 86.0, 174.0, 134.0),
    [] (42.0, 62.0, 134.0, 106.0));

const proc: main is func
  begin
    writeMat(cholesky(m1));
    writeln;
    writeMat(cholesky(m2));
  end func;
```

Output:

```
 5.00000 0.00000 0.00000
 3.00000 3.00000 0.00000
-1.00000 1.00000 3.00000

 4.24264 0.00000 0.00000 0.00000
 5.18545 6.56591 0.00000 0.00000
12.72792 3.04604 1.64974 0.00000
 9.89950 1.62455 1.84971 1.39262
```

Sidef

Translation of: Perl

```
func cholesky(matrix) {
    var chol = matrix.len.of { matrix.len.of(0) };
    matrix.range.each { |row|
        row.range.each { |col|
            var x = matrix[row][col];
            col.range.each { |i|
                x -= (chol[row][i] * chol[col][i]);
            };
            chol[row][col] = (row == col ? x.sqrt : x/chol[col][col]);
        };
    };
    return chol;
};
```

Examples:

```
var example1 = [ [ 25, 15, -5 ],
                 [ 15, 18,  0 ],
                 [ -5,  0, 11 ] ];

say "Example 1:";
cholesky(example1).each { |row|
    say row.map {'%7.4f' % _}.join(' ');
}

var example2 = [ [ 18, 22,  54,  42],
                 [ 22, 70,  86,  62],
                 [ 54, 86, 174, 134],
                 [ 42, 62, 134, 106] ];

say "\nExample 2:";
cholesky(example2).each { |row|
    say row.map {'%7.4f' % _}.join(' ');
}
```

Output:

```
Example 1:
 5.0000  0.0000  0.0000
 3.0000  3.0000  0.0000
-1.0000  1.0000  3.0000

Example 2:
 4.2426  0.0000  0.0000  0.0000
 5.1854  6.5659  0.0000  0.0000
12.7279  3.0460  1.6497  0.0000
 9.8995  1.6246  1.8497  1.3926
```

Smalltalk

```
FloatMatrix>>#cholesky
	| l |
	l := FloatMatrix zero: numRows.
	1 to: numRows do: [:i |
		1 to: i do: [:k | | rowSum lkk factor aki partialSum |
			i = k
				ifTrue: [
					rowSum := (1 to: k - 1) sum: [:j | | lkj |
						lkj := l at: j @ k.
						lkj squared].
					lkk := (self at: k @ k) - rowSum.
					lkk := lkk sqrt.
					l at: k @ k put: lkk]
				ifFalse: [
					factor := l at: k @ k.
					aki := self at: k @ i.
					partialSum := (1 to: k - 1) sum: [:j | | ljk lji |
						lji := l at: j @ i.
						ljk := l at: j @ k.
						lji * ljk].
					l at: k @ i put: aki - partialSum * factor reciprocal]]].
	^l
```

Tcl

Translation of: Java

```
proc cholesky a {
    set m [llength $a]
    set n [llength [lindex $a 0]]
    set l [lrepeat $m [lrepeat $n 0.0]]
    for {set i 0} {$i < $m} {incr i} {
        for {set k 0} {$k < $i+1} {incr k} {
            set sum 0.0
            for {set j 0} {$j < $k} {incr j} {
                set sum [expr {$sum + [lindex $l $i $j] * [lindex $l $k $j]}]
            }
            lset l $i $k [expr {
                $i == $k
                ? sqrt([lindex $a $i $i] - $sum)
                : (1.0 / [lindex $l $k $k] * ([lindex $a $i $k] - $sum))
            }]
```

```
        }
    }
    return $l
}
```

Demonstration code:

```
set test1 {
    {25 15 -5}
    {15 18  0}
    {-5  0 11}
}
puts [cholesky $test1]
set test2 {
    {18 22  54  42}
    {22 70  86  62}
    {54 86 174 134}
    {42 62 134 106}
}
puts [cholesky $test2]
```

Output:

```
{5.0 0.0 0.0} {3.0 3.0 0.0} {-1.0 1.0 3.0}
{4.242640687119285 0.0 0.0 0.0} {5.185449728701349 6.565905201197403 0.0 0.0} {12.727922061357857 3.0460384954008553 1.6497422479090704 0
```

VBA

This function returns the lower Cholesky decomposition of a square matrix fed to it. It does not check for positive semi-definiteness, although it does check for squareness. It assumes that `Option Base 0` is set, and thus the matrix entry indices need to be adjusted if Base is set to 1. It also assumes a matrix of size less than 256x256. To handle larger matrices, change all `Byte`-type variables to `Long`. It takes the square matrix range as an input, and can be implemented as an array function on the same sized square range of cells as output. For example, if the matrix is in cells A1:E5, highlighting cells A10:E14, typing "`=Cholesky(A1:E5)`" and htting `Ctrl-Shift-Enter` will populate the target cells with the lower Cholesky decomposition.

```
Function Cholesky(Mat As Range) As Variant

Dim A() As Double, L() As Double, sum As Double, sum2 As Double
Dim m As Byte, i As Byte, j As Byte, k As Byte

'Ensure matrix is square
    If Mat.Rows.Count <> Mat.Columns.Count Then
        MsgBox ("Correlation matrix is not square")
        Exit Function
    End If

    m = Mat.Rows.Count

'Initialize and populate matrix A of values and matrix L which will be the lower Cholesky
    ReDim A(0 To m - 1, 0 To m - 1)
    ReDim L(0 To m - 1, 0 To m - 1)
    For i = 0 To m - 1
        For j = 0 To m - 1
            A(i, j) = Mat(i + 1, j + 1).Value2
            L(i, j) = 0
        Next j
    Next i

'Handle the simple cases explicitly to save time
    Select Case m
        Case Is = 1
            L(0, 0) = Sqr(A(0, 0))

        Case Is = 2
            L(0, 0) = Sqr(A(0, 0))
            L(1, 0) = A(1, 0) / L(0, 0)
            L(1, 1) = Sqr(A(1, 1) - L(1, 0) * L(1, 0))

        Case Else
            L(0, 0) = Sqr(A(0, 0))
            L(1, 0) = A(1, 0) / L(0, 0)
            L(1, 1) = Sqr(A(1, 1) - L(1, 0) * L(1, 0))
            For i = 2 To m - 1
                sum2 = 0
                For k = 0 To i - 1
                    sum = 0
```

```
                For j = 0 To k
                    sum = sum + L(i, j) * L(k, j)
                Next j
                L(i, k) = (A(i, k) - sum) / L(k, k)
                sum2 = sum2 + L(i, k) * L(i, k)
            Next k
            L(i, i) = Sqr(A(i, i) - sum2)
        Next i
    End Select
    Cholesky = L
End Function
```

zkl

Translation of: C

```
fcn cholesky(mat){
   rows:=mat.len();
   r:=(0).pump(rows,List().write, (0).pump(rows,List,0.0).copy); // matrix of zeros
   foreach i,j in (rows,i+1){
      s:=(0).reduce(j,'wrap(s,k){ s + r[i][k]*r[j][k] },0.0);
      r[i][j]=( if(i==j)(mat[i][i] - s).sqrt()
                else    1.0/r[j][j]*(mat[i][j] - s) );
   }
   r
}
```

```
ex1:=L( L(25.0,15.0,-5.0), L(15.0,18.0,0.0), L(-5.0,0.0,11.0) );
printM(cholesky(ex1));
println("-----------------");
ex2:=L( L(18.0, 22.0,  54.0,  42.0,),
        L(22.0, 70.0,  86.0,  62.0,),
        L(54.0, 86.0, 174.0, 134.0,),
        L(42.0, 62.0, 134.0, 106.0,) );
printM(cholesky(ex2));
```

```
fcn printM(m){ m.pump(Console.println,rowFmt) }
fcn rowFmt(row){ ("%9.5f "*row.len()).fmt(row.xplode()) }
```

Output:

```
  5.00000   0.00000   0.00000
  3.00000   3.00000   0.00000
 -1.00000   1.00000   3.00000
-----------------
  4.24264   0.00000   0.00000   0.00000
  5.18545   6.56591   0.00000   0.00000
 12.72792   3.04604   1.64974   0.00000
  9.89949   1.62455   1.84971   1.39262
```


Categories: Programming Tasks | Matrices | Ada | ALGOL 68 | BBC BASIC | C | Common Lisp | D | DWScript | Fantom | Fortran | Go | Haskell | Haskell examples needing attention | Examples needing attention | Icon | Unicon | J | Java | Jq | Julia | Maple | Mathematica | MATLAB | Octave | Maxima | Nim | Objeck | OCaml | Pascal | Perl | Perl 6 | PicoLisp | PL/I | Python | Q | R | Racket | REXX | Ruby | Scala | Seed7 | Sidef | Smalltalk | Tcl | VBA | Zkl