James D. McCaffrey
Software Research, Development, Testing, and Education

Lower-Upper Matrix Decomposition Implementation in C#

Posted on August 3, 2011

In my last entry I described what lower-upper matrix decomposition is and how it can be used to find the inverse of a matrix. In this entry I give an implementation of lower-upper decomposition in C#. Here's how the method could be called:

```
double[][] m = MatrixCreate(4, 4);
m[0][0] = 8.0; m[0][1] = 6.0; m[0][2] = 4.0; m[0][3] = 2.0;
m[1][0] = 1.0; m[1][1] = 5.0; m[1][2] = 3.0; m[1][3] = 7.0;
m[2][0] = 6.0; m[2][1] = 8.0; m[2][2] = 2.0; m[2][3] = 4.0;
m[3][0] = 9.0; m[3][1] = 3.0; m[3][2] = 5.0; m[3][3] = 1.0;

Console.WriteLine("Original matrix is:");
Console.WriteLine(MatrixAsString(m));

int[] indx = new int[4];
double d = -9; // dummy value
double[][] lum = MatrixDecomposition(m, indx, out d);

Console.WriteLine("Lower and upper stored in a single matrix are:");
Console.WriteLine(MatrixAsString(lum));
```

If this code were run the result would be:

```
Original matrix is:
 8.000   6.000   4.000   2.000
 1.000   5.000   3.000   7.000
 6.000   8.000   2.000   4.000
 9.000   3.000   5.000   1.000

Lower and upper stored in a single matrix are:
 9.000   3.000   5.000   1.000
 0.667   6.000  -1.333   3.333
 0.111   0.778   3.481   4.296
 0.889   0.556   0.085  -1.106
```

The matrix decomposition method is named MatrixDecomposition. I used an algorithm from the well-known book "Numerical Recipes in C". First I create a matrix m using a helper named MatrixCreate which just allocates space. Then I assign some values to m. The MatrixAsString is another helper method. The tricky part is the use of the indx array and the d variable which I'll come back to in a moment. The result of MatrixDecomposition is not two matrices (the lower and the upper) but rather a single matrix which stores both the upper and lower matrices in a clever way. The upper part in this example is:

```
 9.000   3.000   5.000   1.000
 0.000   6.000  -1.333   3.333
```

```
  0.000   0.000   3.481   4.296
  0.000   0.000   0.000  -1.106
```

however the lower part, which always has all 1.0s on the diagonal is:

```
  1.000   0.000   0.000   0.000
  0.667   1.000   0.000   0.000
  0.111   0.778   1.000   0.000
  0.889   0.556   0.085   1.000
```

In other words the 1.0s for the lower are not explicitly stored in the result. Additionally, the result of the MatrixDecomposition methods is permuted — the order of the rows has been changed. The way in which the rows have been reordered is stored in the indx array which is an out parameter. The out d parameter can be used later to find the determinant of the original matrix. In other words, the result of MatrixDecomposition is a single matrix which stores both the lower and upper decomposition of a matrix in row-permuted order, where the 1.0s on the diagonal of the lower part are implied, and where the information necessary to put the rows into the correct order is contained in array indx, and where the out d parameter can be used for other purposes. I'll explain more in my next post. Here's the code:

```
static double[][] MatrixCreate(int rows, int cols)
{
 double[][] result = new double[rows][];
 for (int i = 0; i < rows; ++i) { result[i] = new double[cols]; }

 for (int i = 0; i < rows; ++i)
  for (int j = 0; j < cols; ++j)
   result[i][j] = 0.0; // not really necessary.

 return result;
}

static string MatrixAsString(double[][] matrix)
{
 string s = "";
 for (int i = 0; i < matrix.Length; ++i)
 {
  for (int j = 0; j < matrix[i].Length; ++j)
  {
   s += matrix[i][j].ToString("F3").PadLeft(10) + " ";
  }
  s += Environment.NewLine;
 }
 return s;
} // MatrixAsString

static double[][] MatrixDecomposition(double[][] matrix, int[] indx,
 out double d)
{
 // see "Numerical Recipes in C", 2nd ed., pp. 46-47
 // implements Crout's method with partial pivoting
```

```
int rows = matrix.Length;
int cols = matrix[0].Length;
if (rows != cols)
 throw new Exception("Attempt to MatrixDecomposition a
  non-square mattrix");

int n = rows; // to sync with book notation
int imax = 0; //
double big = 0.0; double temp = 0.0; double sum = 0.0;
double[] vv = new double[n];
d = 1.0; // an out ref parameter

double[][] result = MatrixDuplicate(matrix);
// make a copy of the input array

for (int i = 0; i < n; ++i)
{
 big = 0.0;
 for (int j = 0; j < n; ++j)
 {
  temp = Math.Abs(result[i][j]); // kind of wacky
  if (temp > big)
   big = temp;
 }
 if (big == 0.0)
  return null; // consider throwing an Exception instead
 vv[i] = 1.0 / big;
}

for (int j = 0; j < n; ++j) // each column
{
 for (int i = 0; i < j; ++i)
 {
  sum = result[i][j];
  for (int k = 0; k < i; ++k) { sum -= result[i][k] * result[k][j]; }
  result[i][j] = sum;
 } // i

 big = 0.0;
 for (int i = j; i < n; ++i)
 {
  sum = result[i][j];
  for (int k = 0; k < j; ++k) { sum -= result[i][k] * result[k][j]; }
  result[i][j] = sum;
  temp = vv[i] * Math.Abs(sum);
  if (temp >= big)
  {
   big = temp;
   imax = i;
```

```
      }
    } // i

    if (j != imax)
    {
      for (int k = 0; k < n; ++k)
      {
        temp = result[imax][k];
        result[imax][k] = result[j][k];
        result[j][k] = temp;
      }
      d = -d; // toggle the sign
      vv[imax] = vv[j];
    }
    indx[j] = imax;
    //Console.WriteLine("Setting indx[" + j + "] to " + imax);
    if (result[j][j] == 0.0)
      result[j][j] = 1.0e-20;

    if (j != n - 1)
    {
      temp = 1.0 / result[j][j];
      for (int i = j + 1; i < n; ++i) { result[i][j] *= temp; }
    }

  } // j (each column)

  return result;
} // MatrixDecomposition

static double[][] MatrixDuplicate(double[][] matrix)
{
  double[][] result = new double[matrix.Length][];
  int cols = matrix[0].Length; // assume all columns have equal size
  for (int i = 0; i < result.Length; ++i)
    result[i] = new double[cols];
  for (int i = 0; i < matrix.Length; ++i)
    for (int j = 0; j < matrix[i].Length; ++j)
      result[i][j] = matrix[i][j];
  return result;
}
```


Related

Matrix Inversion in C# using Decomposition
In "Machine Learning"

Unscrambling a Lower-Upper Matrix Decomposition
In "Machine Learning"

Lower-Upper Matrix Decomposition in C#
In "Machine Learning"

This entry was posted in Machine Learning. Bookmark the permalink.

James D. McCaffrey