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

Matrix Inversion in C# using Decomposition

Posted on August 15, 2011

It turns out that a naive approach to finding the inverse of a matrix is usually inefficient. The standard approach is to break down the matrix to be inverted into two matrices (lower and upper) and then use back substitution. For example:

```
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));

double[][] inv = MatrixInverse(m);
Console.WriteLine("The inverse is:");
Console.WriteLine(MatrixAsString(inv));

double[][] prod = MatrixProduct(m, inv);
Console.WriteLine("Product of m * inv is:");
Console.WriteLine(MatrixAsString(prod));
```

The well-known book "Numerical Recipes in C" has all the details and I refactored that code into C#. Here's the code:

```
static double[][] MatrixInverse(double[][] matrix)
{
 // use clever matrix decomposition technique.
 // returns null on error.
 int r = matrix.Length;
 int c = matrix[0].Length;
 if (r != c)
  throw new Exception("Attempt to MatrixInverse a non-square mattrix");

 int n = r;
 double[][] result = MatrixCreate(n, n);
 double[] col = new double[n];
 double[] x = new double[n];

 int[] indx = new int[n];
 double d;
 double[][] luMatrix = MatrixDecomposition(matrix, indx, out d);

 if (luMatrix == null) return null;
```

```
  for (int j = 0; j < n; ++j)
  {
    for (int i = 0; i < n; ++i) { col[i] = 0.0; }
    col[j] = 1.0;
    x = MatrixBackSub(luMatrix, indx, col);
    for (int i = 0; i < n; ++i) { result[i][j] = x[i]; }
  }
  return result;
} // MatrixInverse

static double[][] MatrixDecomposition(double[][] matrix,
  int[] indx, out double d)
{
  // see earlier blog post
}

static double[] MatrixBackSub(double[][] luMatrix,
  int[] indx, double[] b)
{
  int rows = luMatrix.Length;
  int cols = luMatrix[0].Length;
  if (rows != cols)
    throw new Exception("Non-square LU mattrix");

  int ii = 0; int ip = 0;
  int n = b.Length;
  double sum = 0.0;

  double[] x = new double[b.Length];
  b.CopyTo(x, 0);

  for (int i = 0; i < n; ++i)
  {
    ip = indx[i];
    sum = x[ip];
    x[ip] = x[i]; //
    if (ii == 0)
    {
      for (int j = ii; j <= i – 1; ++j)
        { sum -= luMatrix[i][j] * x[j]; }
    }
    else if (sum == 0.0)
    {
      ii = i;
    }
    x[i] = sum;
  } // i

  for (int i = n – 1; i >= 0; –i)
  {
```

```
    sum = x[i];
    for (int j = i + 1; j < n; ++j)
      { sum -= luMatrix[i][j] * x[j]; }
    x[i] = sum / luMatrix[i][i];
  }
  return x;
} // MatrixBackSub
```


Related

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

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

Logistic Regression
In "Machine Learning"

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

James D. McCaffrey