

Matrix used in linear algebra

/*

Copyright (c) 2010 [James Craig](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.*/

```
#region Usings
using System;
```

```

using System.Text;
using System.Xml.Serialization;
#endregion

namespace Utilities.Math
{
    ///
    /// Matrix used in linear algebra
    ///
    [Serializable()]
    public class Matrix
    {
        #region Constructor
        ///
        /// Constructor
        ///
        /// Width of the matrix /// Height of the matrix public
        Matrix(int Width,int Height)
        {
            _Width = Width;
            _Height = Height;
            _Values = new double[Width, Height];
        }
        #endregion

        #region Public Properties
        ///
        /// Width of the matrix
        ///
        [XmlElement]
        public int Width
        {
            get { return _Width; }
            set { _Width = value; _Values = new double[Width, Height]; }
        }
    }
}

```

```

///

/// Height of the matrix
///
[XmlElement]
public int Height
{
get { return _Height; }
set { _Height = value; _Values = new double[Width, Height]; }
}
///

/// Sets the values of the matrix
///
/// X position /// Y position /// the value at a point in the
matrix
public double this[int X, int Y]
{
get
{
if (X < _Width && X >= 0 && Y < _Height && Y >= 0)
{
return _Values[X, Y];
}
throw new Exception("Index out of bounds");
}

set
{
if (X < _Width && X >= 0 && Y < _Height && Y >= 0)
{
_Values[X, Y] = value;
return;
}
throw new Exception("Index out of bounds");
}
}
}

```

```

///

/// Values for the matrix
///
[XmlElement]
public double[,] Values
{
get { return _Values; }
set { _Values = value; }
}
#endregion

#region Private Variables
private int _Width = 1;
private int _Height = 1;
private double[,] _Values = null;
#endregion

#region Operators
public static bool operator ==(Matrix M1, Matrix M2)
{
if (M1.Width != M2.Width || M1.Height != M2.Height)
return false;
for (int x = 0; x <= M1.Width; ++x) { for (int y = 0; y <=
M1.Height; ++y) { if (M1[x, y] != M2[x, y]) return false; } }
return true; } public static bool operator !=(Matrix M1,
Matrix M2) { return !(M1 == M2); } public static Matrix
operator +(Matrix M1, Matrix M2) { if (M1.Width != M2.Width ||
M1.Height != M2.Height) throw new ArgumentException("Both
matrices must be the same dimensions."); Matrix TempMatrix =
new Matrix(M1.Width, M1.Height); for (int x = 0; x < M1.Width;
++x) { for (int y = 0; y < M1.Height; ++y) { TempMatrix[x, y]
= M1[x, y] + M2[x, y]; } } return TempMatrix; } public static
Matrix operator -(Matrix M1, Matrix M2) { if (M1.Width !=
M2.Width || M1.Height != M2.Height) throw new
ArgumentException("Both matrices must be the same
dimensions."); Matrix TempMatrix = new Matrix(M1.Width,
M1.Height); for (int x = 0; x < M1.Width; ++x) { for (int y =

```

```

0; y < M1.Height; ++y) { TempMatrix[x, y] = M1[x, y] - M2[x,
y]; } } return TempMatrix; } public static Matrix operator -
(Matrix M1) { Matrix TempMatrix = new Matrix(M1.Width,
M1.Height); for (int x = 0; x < M1.Width; ++x) { for (int y =
0; y < M1.Height; ++y) { TempMatrix[x, y] = -M1[x, y]; } }
return TempMatrix; } public static Matrix operator *(Matrix
M1, Matrix M2) { if (M1.Width != M2.Width || M1.Height !=
M2.Height) throw new ArgumentException("Dimensions for the
matrices are incorrect."); Matrix TempMatrix = new
Matrix(M2.Width, M1.Height); for (int x = 0; x < M2.Width;
++x) { for (int y = 0; y < M1.Height; ++y) {
TempMatrix[x,y]=0.0; for (int i = 0; i < M1.Width; ++i) { for
(int j = 0; j < M2.Height; ++j) { TempMatrix[x, y] += (M1[i,
y] * M2[x, j]); } } } } return TempMatrix; } public static
Matrix operator *(Matrix M1,double D) { Matrix TempMatrix =
new Matrix(M1.Width, M1.Height); for (int x = 0; x < M1.Width;
++x) { for (int y = 0; y < M1.Height; ++y) { TempMatrix[x, y]
= M1[x, y] * D; } } return TempMatrix; } public static Matrix
operator *(double D, Matrix M1) { Matrix TempMatrix = new
Matrix(M1.Width, M1.Height); for (int x = 0; x < M1.Width;
++x) { for (int y = 0; y < M1.Height; ++y) { TempMatrix[x, y]
= M1[x, y] * D; } } return TempMatrix; } public static Matrix
operator /(Matrix M1, double D) { return M1 * (1 / D); }
public static Matrix operator /(double D, Matrix M1) { return
M1 * (1 / D); } #endregion #region Public Overridden Functions
public override bool Equals(object obj) { if (obj is Matrix) {
return this == (Matrix)obj; } return false; } public override
int GetHashCode() { double Hash=0; for (int x = 0; x < Width;
++x) { for (int y = 0; y < Height; ++y) { Hash += this[x, y];
} } return (int)Hash; } public override string ToString() {
StringBuilder Builder = new StringBuilder(); string Seperator
= ""; Builder.Append("{ " + System.Environment.NewLine); for
(int x = 0; x < Width; ++x) { Builder.Append("{"); for (int y
= 0; y < Height; ++y) { Builder.Append(Seperator + this[x,
y]); Seperator = ","; } Builder.Append("} " +
System.Environment.NewLine); Seperator = ""; }
Builder.Append("}"); return Builder.ToString(); } #endregion

```

```

#region Public Functions ///
/// Transposes the matrix
///

/// Returns a new transposed matrix
public Matrix Transpose()
{
Matrix TempValues = new Matrix(Height, Width);
for (int x = 0; x < Width; ++x) { for (int y = 0; y < Height;
++y) { TempValues[y, x] = _Values[x, y]; } } return
TempValues; } ///
/// Gets the determinant of a square matrix
///

/// The determinant of a square matrix
public double Determinant()
{
if (Width != Height)
throw new Exception("The determinant can not be calculated for
a non square matrix");
if (Width == 2)
{
return (this[0, 0] * this[1, 1]) - (this[0, 1] * this[1, 0]);
}
double Answer = 0.0;
for (int x = 0; x < Width; ++x) { Matrix TempMatrix = new
Matrix(Width - 1, Height - 1); int WidthCounter = 0; for (int
y = 0; y < Width; ++y) { if (y != x) { for (int z = 1; z <
Height; ++z) { TempMatrix[WidthCounter, z - 1] = this[y, z]; }
++WidthCounter; } } if (x % 2 == 0) { Answer +=
TempMatrix.Determinant(); } else { Answer -=
TempMatrix.Determinant(); } } return Answer; } #endregion } }
[/csharp]

```