Tutorial | fairmat

Fairmat numerical API introduction (for matlab users)

Revision # 5, updated to Fairmat 1.6.x API

1 New methods from 1.6.1

- Vector.MovingAverage(...
- Sparse Indexing var a = b[new int[]1,3,7]; b[new int[]1, 5]= new Vector(2);
- Correlation Analysis var m= new Matrix(100,3); var b=m.Correlation();

2 Introduction

The Fairmat API contains a series of classes which deal with numerical algorithm matrices and algebra. This tutorial shows some of the features of the numerical classes and puts a particular emphasis on the similarities and differences with the Matlab language.

This tutorial does not contain the complete list of available methods and operations. For a complete reference of the Fairmat API go to `http://www.fairmat.com/documentation/fairmatapi/namespaces.htm`.

3 Matrix and Vector Classes

The Fairmat API contains Vector and Matrices[1] class, the main difference between Matlab matrices and C# matrices is indexing. Matlab matrix indices are one-based, meaning that indices start with the value one, while C# matrix and array indices are in general zero-based.

In the following sections, operations than can be done with matrices are shown in both implementations.

[1]Vector, Matrix and Matrix3 are defined in the assembly DVPLI.dll, namespace DVPLI.

3.1 Matrices initialization

Matlab	C# + Fairmat library
a = **zeros**(1,2);	`var a = new Matrix(1,2);`
b = **ones**(1,2);	`var b = Matrix.Ones(1,2);`
c = **zeros**(10);	`var c = new Matrix(10);`
d = 5***ones**(10,20);	`var d = 5*Matrix.Ones(10,20);`
e = [1, 2, 3, 4];	`var e = (Matrix)new double[]{1,2,3,4};`
f = [1 2 ; 3 4];	`var f = (Matrix)new double[,]{{1,2},{3,4}};`
g= **eye**(5);	`var g = Matrix.Identity(5);`
h = [f f ; f f];	`var h = new Matrix[][]{new[]{f,f}, new[]{f,f}}`
i = **reshape**(e, 2, 2);	`var i = e.Reshape(2, 2);`
l = **sum**(f);	`var l = f.ColumnSum();`

3.2 Popular matrices operations

Matlab	C# + Fairmat library
m = **max**(f);	`m = f.Max();`
n = **mean**(f);	`n = f.Means();`
o = **sum**(f(:));	`o = f.Sum();`
p = **mean**(f(:));	`p = f.Mean();`
q = f.^2;	`q = Matrix.Pow(f, 2);`
q2 = f^2;	`q2 = Matrix.PowM(f,2);`
q3 = **expm**(f);	`q3 = Matrix.ExpM(f);`
r = **sqrt**(f);	`r = Matrix.Sqrt(f);`
s = **log**(f);	`s = Matrix.Log(f);`
t = **exp**(f);	`t = Matrix.Exp(f);`
u = **det**(f);	`u = f.Determinant();`
v = **inv**(f);	`v = f.Inverse();`

3.3 Vectors initialization and operations

In a similar way, the vector class easy allows to work on vectors.

Matlab	C# + Fairmat library
a = [1 2];	`var a = new Vector(){1, 2};`
a = **linspace**(10,20,200);	`var a = Vector.Linspace(10,20,200);`
b = **sum**(a)	`double b = a.Sum();`
c = **cumsum**(a)	`var c = a.CumSum();`
d = a[10:100];	`var d = a[Range.New(9,99)];`

3.4 Matrices transposition

Matlab	C# + Fairmat library
a = **zeros**(1,2);	`var a = new Matrix(1,2);`
a = a';	`a = a.T;`

3.5 Matrices arithmetics

Matlab	C# + Fairmat library
a = **ones**(2,1);	`var a = Matrix.Ones(2,1);`
b = **ones**(1,2)	`var b = Matrix.Ones(1,2);`
d = a*b − 3;	`var d = a*b - 3;`

3.6 Operations on range of indices

One of the point of strength of Fairmat Matrix API is that the classes allows easy access to range of indices or sub-section of the matrices.

Matlab	C# + Fairmat library
a = **ones**(3,3)	`var a = Matrix.Ones(3,3);`
b = a(:,1)	`var b = a[Range.All, 0];`
c = **zeros**(3,4)	`var c = new Matrix(3,4);`
c(:,1) = b(2,:)	`c[Range.All,0] = b[1,Range.All];`
d = **zeros**(4,4);	`var d = new Matrix(4,4);`
d(1:2,1:2) = **ones**(2,2);	`d[Range.New(0,1),Range.New(0,1)] = Matrix.Ones(2,2);`
e = **zeros**(10,1);	`var e = new Vector(10);`
e(1:5) = 10.45;	`e[Range.New(0,4)] = (Vector)10.45;`

3.7 Algebra

The API provides some basic linear algebra algorithms.

Matlab	C# + Fairmat library
a = **ones**(3,3)	`var a = Matrix.Ones(3,3);`
d = **det**(a);	`double d = a.Determinant();`
e = **chol**(a);	`var e = a.Cholesky();`
f = **inv**(a);	`var f = a.Inverse();`

3.8 Linear Regression

Linear regression classes are available in the Assembly DVPLUtils.dll, namespace Fairmat.Statistics.

Matlab	C# + Fairmat library
c=a\b	`Vector c = LinearRegression.Solve(a,b);`

3.9 Statistics

3.10 Descriptive Statistics

The API provides some basic descriptive statistics.

Matlab	C# + Fairmat library
a = **ones**(3,3)	`var a = Matrix.Ones(3,3);`
m = **mean**(a);	`Vector m = a.Mean();`
cov = **cov**(a);	`var cov = a.Covariance();`
corr = corr(a);	`var corr = a.Correlation();`

3.11 3-Dimensional Matrices

We provide also a basic 3-dimension matrix.

Matlab	C# + Fairmat library
	`//a is a 2x3x4 matrix`
a = **ones**(2,3,4)	`var a = Matrix3.Ones(2,3,4);`
	`//two-dim sub matrices`
b = a[:,:,1]	`var b = a[Range.All, Range.All,0];`
c = a[:,1,:]	`var c = a[Range.All,0, Range.All];`
d = a[0,:,:]	`var d = a[0,Range.All, Range.All];`
e = a[:,1,2]	`var e = a[Range.All,0,1];`

4 Numerical integration

Integration is performed by the class `Fairmat.Math.Integrate` which takes an `IIntegrable` object or an `Integrand` delegate as constructor input. See the example below:

```
//1) Includes the integrand function in a class implementing IIntegrable
public class IntegrandClass: IIntegrable
{
  public double IntegrandFunc(double x)
 {
    return Math.Sin(x)*Math.Pow(x,3);
 }
}
//2) Instance the Integrate class
...
Integrate z= new Integrate(IntegrandClass);

//3) use one of the available integration methods in order to integrate from a
to b
var int1=z.AdaptLobatto(a, b);
var int2=z.Rectangle(a,b,20);
var int2=z.Romberg(a,b)
```

5 Mathematical Programming

The Fairmat library contains also some implementation of optimization algorithms suited for small-size nonlinear mathematical programming. problems.[2]

We provide both an object oriented framework and an imperative (delegate based) framework. In the object oriented version, the mathematical programming problem is defined by a class like in the following example:

```
public class SimpleProblem: IOptimizationProblem
{
```

[2]The Mathematical programming algorithms are contained in the assembly Fairmat.Optimization.dll, namespace Fairmat.Optimization.

```
    public double Obj(DVPLI.Vector x)
    {
        return (x - 2.0).Scalar(x - 2.0);
    }

    //implement (is applicable) analytical gradient
    public DVPLI.Vector Grad(DVPLI.Vector x)
    {
        throw new NotImplementedException();
    }

    public Bounds Bounds
    {
        get
        {
            Bounds b= new Bounds();
            b.Lb =(Vector) new double[] { 0, 0,0,0,0};
            b.Ub = (Vector)new double[] { 100, 100, 100, 100, 100 };
            return b;
        }
    }

    //Defines the constraint  Ax<=b
    public LinearConstraints LinearIneqConstraints
    {
        get
        {
            LinearConstraints l = new LinearConstraints();
            Matrix A = new Matrix(1, 5);
            Vector b = new Vector(5);
            A[0, 0] = -1;
            A[0, 1] = +1;
            l.A = A;
            l.b = b;
            return l;
        }
    }

    public bool HasNonLinearConstraints
    {
        get { return false;}
    }

    //feasible region is defined by x | G(x)<=0
    public DVPLI.Vector G(DVPLI.Vector x)
    {
        throw new NotImplementedException();
    }
}
```

After the problem is defined you can find the optimal solution in the following way:

```
//Instantiate a Quasi Newton local optimization algorithm
IOptimizationAlgorithm algorithm= new SteepestDescent();

OptimizationSettings settings = new OptimizationSettings();


settings.MaxIter=50; // maximum number of iteration allowed
settings.Verbosity = 1; // positive integer values print debug info
settings.epsilon = 10e-4;// tolerance

//optional starting point
Vector x0 = (Vector)new double[] { 10, 30,50,60 };
var solution = algorithm.Minimize(new SimpleProblem(),settings,x0);
Console.WriteLine(solution);
```

In addition we offer a delegate based API which can be used for simpler problems defined only by an objective function:

```
Vector x0 = (Vector)new double[] { 10, 30,50,60 };
var solution = Fairmat.Optimization.Helper.Minimize(new ObjFunc(Func),x0);
Console.WriteLine(solution);
...
double Func(Vector x){
return ...
}
```