The SniffLib Manual

Daniel P. Dougherty

Beta Release

SniffLib™ "I smell something brewing..."

The SniffLib Manual

Daniel P. Dougherty

Preface

The Sniff numerical library provides honest-to-goodness N-dimensional array construction and manipulation along with standard linear algebra functionality as well as statistics and computational routines in the JAVA language. This is an open source project and input from open source developers is needed and encouraged.

Contents

Chapter 1

Notes to Developers

1.1 Background and basic policies

SniffLib came into being when the original author was developing mathematical models of the olfactory system and needed a JAVA-based numerical library. Now that the library is an open–source project it may be more appropriate if SniffLib refers to "<u>S</u>tandard <u>N</u>umerical <u>I</u>nter<u>f</u>aces and <u>F</u>unctions <u>Lib</u>rary" as the library is meant to provide not only numerical routines but generic interfaces and abstract classes typically required in numerical calculations. For example, the library provides the interface *datatypes.NamedParameters* which provides the getParam and setParam methods. Numerical classes implementing the *datatypes.NamedParameters* interface might have methods for parameter optimization, function evaluation, parameter sensitivity analysis etc.

The reason for implementing the library in JAVA is that the new JVM allows competitive speed and multi-threaded class methods may have an advantage over classical computational tools such as MATLAB. The SNIFFLIB$^{\text{TM}}$library also attempts to use the standard JAVA API's and in particular the Collections API whenever possible. The judicious use of streams can also benefit numerical computations especially when datasets become large. Several computational products, R and Splus for example, save variables to files (e.g. in the .data directory in Splus). This kind of enforced usage of file-based I/O is hidden from the user and is greatly needed for computations in which all the data can not be read into RAM. It also makes sense for statistical datasets that are often queried by SQL-like commands to create new data sets. It is my experience however that this strategy is often not required for most numerical algorithms. Thus rather than "enforce policy" about how data is used it would be better (IMHO) to allow the algorithm designer to decide how to manage memory. This includes the omnipresent issue of memory management. Dynamic memory allocation and reallocation is often required in numerical computations and in general programming. However automatic dynamic memory allocation, often a feature of numerical libraries implementing mathematical matrices, is a constant source of bugs and can encourage lazy programming practices that lead to weak performance. The idea is to not "enforce policy" regarding memory management. People wanting automatic memory management can use some of the softwares mentioned above. SNIFFLIB$^{\text{TM}}$developers should provided memory management schemes, especially general ones, that algorithm designers can use or not use depending on the application.

SNIFFLIB$^{\text{TM}}$development should not, however, eschew all convenience tactics. The most consistent need for convenience by numerical program end-users comes in the form of overloaded constructors, public static methods, and public instance methods. For example, sometimes one might want to do DblMatrix.add(DblMatrix A) but DblMatrix.add(int A) is also very commonly desired. Developers should recognize when users might want to use primitive types as input rather than say the *datatypes.DblMatrix* class. Recognizing and creating convenience constructors which use default values is also a good tactic.

1.2 Preferred Programming Style

Prior to each method, class and interface definition there should be a Javadoc style comment block. For example, consider the following HelloWorld class.

```
import java.io.*;

/**
This class can say hello.
This class does much less than it should but doesn't do any more than it can.
I could go on but I won't.
*/
public class HelloWorld()
{
String message = "Hello world";

public void speak()
{
System.out.println("Hello world.");
}
}
```

Notice that the first line gives a key definition of what the class is about. More details are given on subsequent lines. To create hyperlinks to other classes within the javadoc output use a {@link *package.class*} or {@link *package.class.method*}.

1.3 Graphical User Interface Development

People have their own habits when designing graphical user interfaces. I found that JAVA GUI development can be really treacherous compared to other languages. Here are some guides that I try to use:

1.3.1 Use switch–yard programming to handle action commands

JAVA really, really needs a String–based switch. It would greatly improve maintainability of GUI component code and enforce good coding habits. But apparently Sun's just not going to do that for us anytime soon. However, I feel that the following code which uses the "while true" trick with breaks behaves just as a String-based switch on the String callback might and is very readable, very recognizable as a switch-yard, and therefore maintainable.

```
while (true)//Switch yard starts here
{
        if (callback.equalsIgnoreCase("do_this"))
        {
                System.out.println("Yo");
                break;
        }

        if (callback.equalsIgnoreCase("do_that"))
        {
                System.out.println("YoYo");
                break;
        }

        if (callback.equalsIgnoreCase("do_the_other"))
        {
                System.out.println("YoYoYo");
                break;
        }


        System.out.println("Warning:Unrecognized callback.");
        break;
}
```

Do not use enum switch yards as this is very bad programming! Although one can use all kinds of crazy internal classes and anonymous classes when implementing action listeners I find these practices lead to spaghetti codes of a sort and are very hard to read and therefore maintain. It also is a bad habit to get into when trying to develop re-usable components. In my experience a better way to do it is to always allow the constructors of your GUI components to accept an ActionListener (or other appropriate Listener). Your component should add this Listener to the components the parent GUI component might need to know about. Your component can and should use internal ActionListeners for things the parent component (read user) shouldn't need to deal with. The idea then is to put a switch yard in the parent component which handles all action commands including its own and the relevent ones from its added components. Multiple such switch yards are OK but should be kept a minimum. This kind of centralized brain of the GUI keeps things clean and neat and encourages developers to write clean and neat re-usable GUI components rather than kludging–up a bunch of stuff with trip wires and duct tape ...

Chapter 2

Getting Started

2.1 Downloading and Installing

2.2 Building Your Code

You are strongly encouraged

2.3 Debugging

Chapter 3

Matrix Construction and Manipulation

3.1 Creating Matrices

The class you'll use most in creating and manipulating numbers is *datatypes.DblMatrix*. There are several convenient construction methods for this class. The following code gives two ways to create the same matrix.

```
import datatypes.*;

public class testDbl
{
        public static void main(String[] args)
        {
                //Want to create the matrix
                //
                // [1 2 3;
                //  4 5 6;
                //  7 8 9]
                //
                //

//Method 1: Fill in.
                int[] siz = new int[2];
                siz[0] = 3;
                siz[1] = 3;

                DblMatrix X = new DblMatrix(siz);
        int k = 0;
                for (int i=0;i<X.getN();i++)
                {
X.setDoubleAt(i+1,i);
                }
                X = X.transpose(); //Flip rows an columns.
//Method 2: String representation
DblMatrix X = new DblMatrix("[1 2 3; 4 5 6; 7 8 9]");


}
}
```

MATLAB, Splus and other software have a nice feature that when you type a variable's name and hit return the variable's data is pretty-printed on the screen.

```
>> X = [1 2 3; 4 5 6; 7 8 9]

X =

     1     2     3
     4     5     6
     7     8     9
```

You can get something of the same effect with DblMatrix's *DblMatrix.show* method.

```
DblMatrix X = new DblMatrix("[1 2 3; 4 5 6; 7 8 9]");
X.show();
X.show("X");
```

will produce on the screen

```
[matrix] =
    1.0 2.0 3.0
    4.0 5.0 6.0
    7.0 8.0 9.0

[X] =
    1.0 2.0 3.0
    4.0 5.0 6.0
    7.0 8.0 9.0
```

Notice that the label is by default "matrix" but you can make it anything you want. Using show can be helpful in debugging programs.

3.2 Indexing and Subscripting

There is an *Subscript* class which you may find useful in manipulating the data within a matrix. Subscript arrays can be created to index into each dimension of a *DblMatrix*. Moreover, the *Subscript* class provides the ability to set starting and stopping indices. This allows one to change the range of an index without having to reallocate memory for a completely new *Subscript* array. The following code show how to obtain the lower triangular portion of a matrix and uses this functionality.

```
/**
Get lower triangular portion of a matrix.
*/
```

```
public static DblMatrix tril(DblMatrix X,int diag)
{
Subscript[] subs = new Subscript[2];
subs[0] = new Subscript(DblMatrix.span(0,X.Size[0]-1,X.Size[0]));
subs[1] = new Subscript(new Double(0.0));
subs[0].setStart(0);
subs[0].setStop(X.Size[0]-1);

DblMatrix R = new DblMatrix(X.Size);
int val;

for (int k=0;k<X.Size[1];k++)
{
subs[1].setDoubleAt(new Double(k),0);
val = k+diag;
if (val < X.Size[0])
{
val = Math.max(val,0);
subs[0].setStart(val);
R.setSubMatrix(X.getSubMatrix(subs),subs);
}
}

return(R);
}
```

Chapter 4

Basic Mathematical Operations

All the methods in *java.lang.math* have been overloaded as either static methods in the DlbMatrix class or as instance methods. In some cases the same functionality is granted to both instances and static methods but in these cases the method name might have been changed.

For example to add a DblMatrix A, element by element, to a DblMatrix B (of the same dimension of course) you'd say A.plus(B). But you can also say DblMatrix.add(A,B). Similarly for subtraction you can say either A.minus(B) or DblMatrix.subtract(A,B).

4.0.1 Statistics

The DblMatrix class provides many basic statistics such as the sum, the mean, the standard deviation. These methods typically operate by default down the first (row) dimension. For example DblMatrix.sum(A) returns the column sums of A. An integer second argument will result in the summation down the dimension it specifies. For example, DblMatrix.sum(A,2) gives the row sums.

Chapter 5

Linear Algebra

Nothing to report yet.

Chapter 6

Differential Equations

Need a brief mathematical introduction here.

6.1 Ordinary Differential Equations

The differential equation solver classes provided by SniffLib™ are abstract and must be extended by a user's class which implements the DblMatrix deriv() method. Because the parent ODE solver class implements the *invprobs.NamedParameters* interface the constructor can be used to set parameters for the problem. In the following example the logistic equation is solved. The logistic equation has the form:

$$\frac{dY}{dt} = \alpha \cdot \left(1 - \frac{Y}{K}\right) \cdot Y \tag{6.1}$$

where α and K are parameters. In the following example, the parameters are set in the constructor and gotten in the deriv method. The parameters set in the constructor could be read from a file or some other source but in this example they are set explicitly at instantiation.

```
package integration;
import datatypes.DblMatrix;
import datatypes.ParamSet;

public class Logistic extends OdeSolve
{
        public Logistic(DblMatrix Tstart,DblMatrix Tend,DblMatrix yinit)
        {
                super(Tstart,Tend,yinit);
                this.setParam("alpha",new DblMatrix(new Double(0.1)));
                this.setParam("K",new DblMatrix(new Double(1.0)));
        }

        public Logistic(DblMatrix Tspan,DblMatrix yinit)
        {
                super(Tspan,yinit);
                this.setParam("alpha",new DblMatrix(new Double(0.1)));
                this.setParam("K",new DblMatrix(new Double(1.0)));
        }

        public DblMatrix deriv()
        {

                DblMatrix dYdt;
                DblMatrix alpha = this.getParam("alpha");
                DblMatrix K = this.getParam("K");

                dYdt = alpha.times(DblMatrix.subtract(1,this.Ynow.divideBy(K))).times(this.Ynow);

                return(dYdt);
        }
}
```

Once an ODE class has been instantiated the problem can be solved my invoking the solve() method. The following code demonstrates the solution of the logistic equation on 25 points over the interval [0,10] using 0.1 as an initial condition. The solutions at each of the 25 points are saved to the text file results.dat.

```
import java.io.*;
import integration.*;
import datatypes.*;

public class testode
{
public static void main(String[] args) throws IOException
{

DblMatrix tspan = DblMatrix.span(0,10,25);

Logistic myode = new Logistic(tspan,0.1);
myode.solve();
myode.export("results.dat");
}
}
```

6.2 Stochastic Differential Equations

This is still under development but here is a preliminary example using the stochastic Brusselator.

```
import java.io.*;
import integration.*;
import datatypes.*;

public class testbruss
{
        public static void main(String[] args) throws IOException
        {
                int[] yinitSiz = new int[2];
                yinitSiz[0] = 2;
                yinitSiz[1] = 1;

                DblMatrix yinit = new DblMatrix(yinitSiz);
                yinit.set(new Double(1),0);//Initial X
                yinit.set(new Double(0.1),1);//Initial Y

                DblMatrix tspan = DblMatrix.span(0,40,50);


                StochBruss MYPROB1 = new StochBruss(tspan,yinit,2);
                MYPROB1.Options.Interpolate = "no";
                MYPROB1.Options.Ito2Strat = "yes";
                MYPROB1.Options.Verbose = "no";
                MYPROB1.Options.RelTol = new DblMatrix(new Double(1.e-2));
                MYPROB1.Options.NoStochastic = "no";

                MYPROB1.solve();//Solve the bloody thing already!!

                MYPROB1.export("walk1.txt");
                System.out.println("Numerical integration successful...");
                System.out.println("Tree Dimension:"+MYPROB1.BT[0].getDimension());
        }
}
```

Chapter 7

Model Fitting

Under development.

Chapter 8

Utilities

The *snifflib.util* package provides various utilities which don't fall neatly under any other category.

Index