SonicDaisy - Desktop-based software for surrogate modeling

Technical Documentation

Adem Kikaj

July 28, 2020

Contents

1 Introduction **3**

2 SonicDaisy **4**

3 ProBMoT Parser **16**

4 Latin Hypercube Sampling **18**

5 Fourier Transform and Probability Density Function **19**

6 Data sets Visualization **20**

7 Clus Builder **21**

1 Introduction

This is a technical documentation of the SonicDaisy desktop-based software for automated surrogate modeling.

SonicDaisy is Java-based software with a rich User Interface that is used to ease the surrogate modeling process. The surrogate models in the SonicDaisy are built by ProBMoT [1] only on complete models, thus, to continue building predictive clustering trees by Clus [2].

SonicDaisy is able to understand the attributes of a ProBMoT complete model by opening ProBMoT model file of the type *.pbm* and library file of the type *.pbl*. After the files have been parsed a specific number of simulations can be run (more at ProBMoT Parser).

After having a data set of ProBMoT simulations, SonicDaisy is able to transform the output simulation results (targets) to Fourier Transform (more at Fourier Transform and Probability Density Function) or Probability Density Function (more at Fourier Transform and Probability Density Function). In the case when both transformations are run, the user will have three different data sets where the user can choose which one to run on Clus.

On the process of building a surrogate model, SonicDaisy offers an interactive feature of charts, where the user can see specific types of results like one or multiple simulation results of the original data set or the other transformed data sets (more at Data sets Visualization).

2 SonicDaisy

SonicDaisy is a Java-based software developed using Java 8 and JavaFX. Besides JavaFX elements there is also included a third part library which runs in web browsers and it is JavaScript based known as Charts.js [3]. The main goal of this software is to build surrogate models using ProBMoT [1] and Clus [2].

To run SonicDaisy, make sure that you have installed on your system a JRE not older than the 8th version that can be found at the Oracles official website. After installation of JRE make sure that the $JAVA\_HOME$ path is set up properly.

After everything is set up the user can run SonicDaisy and the main window of the software will appear (see Fig. 1).

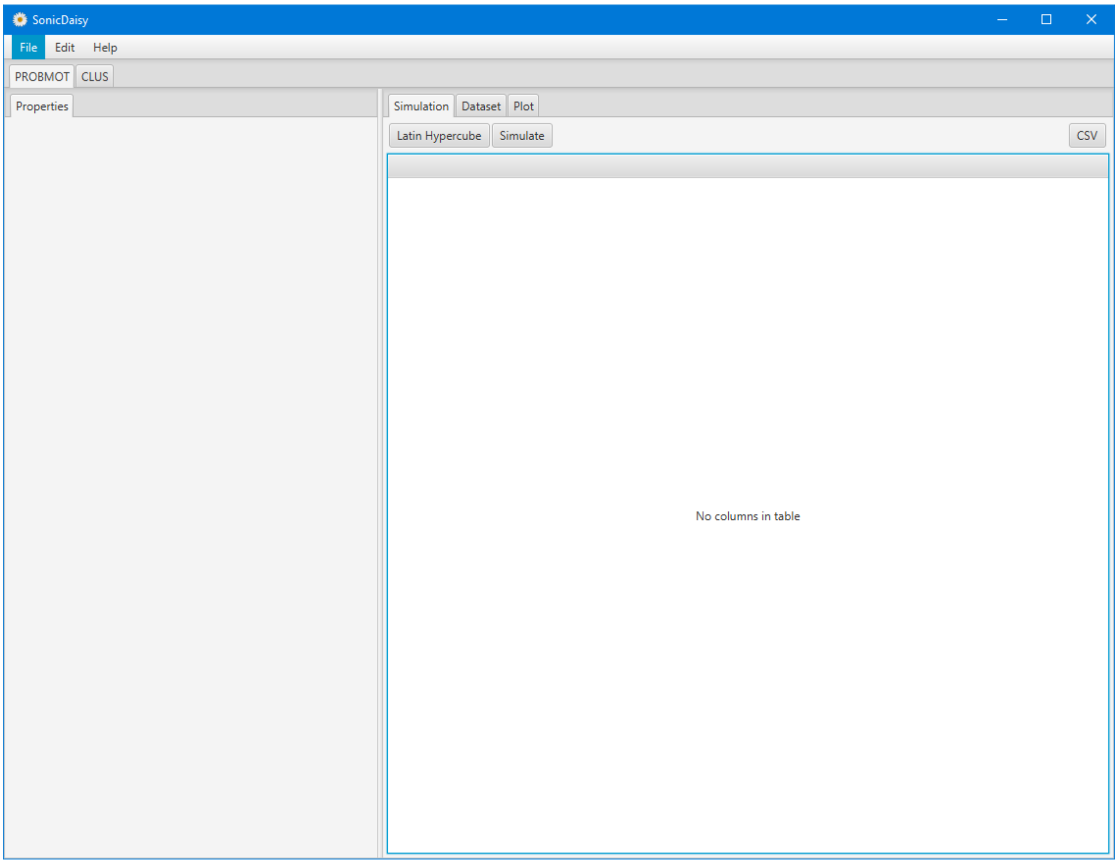

Figure 1: Main window of SonicDaisy.

The main window is composed of two main Tabs named *PROBMOT* and *CLUS*. In the first tab which is selected in Fig. 1, the imported ProBMoT model will be showing its own properties after a successful parsing process. SonicDaisy works only with complete ProBMoT models.

To Parse a ProBMoT model (more about ProBMot parsing at section 3) the user has to import the ProBMoT model and its own corresponding ProBMoT library by going to the menu item *Open* under menu *File* or through shortcut $Ctrl + O$ and select specific *.pbm* and *.pbl* files. This process is shown in Fig. 2.

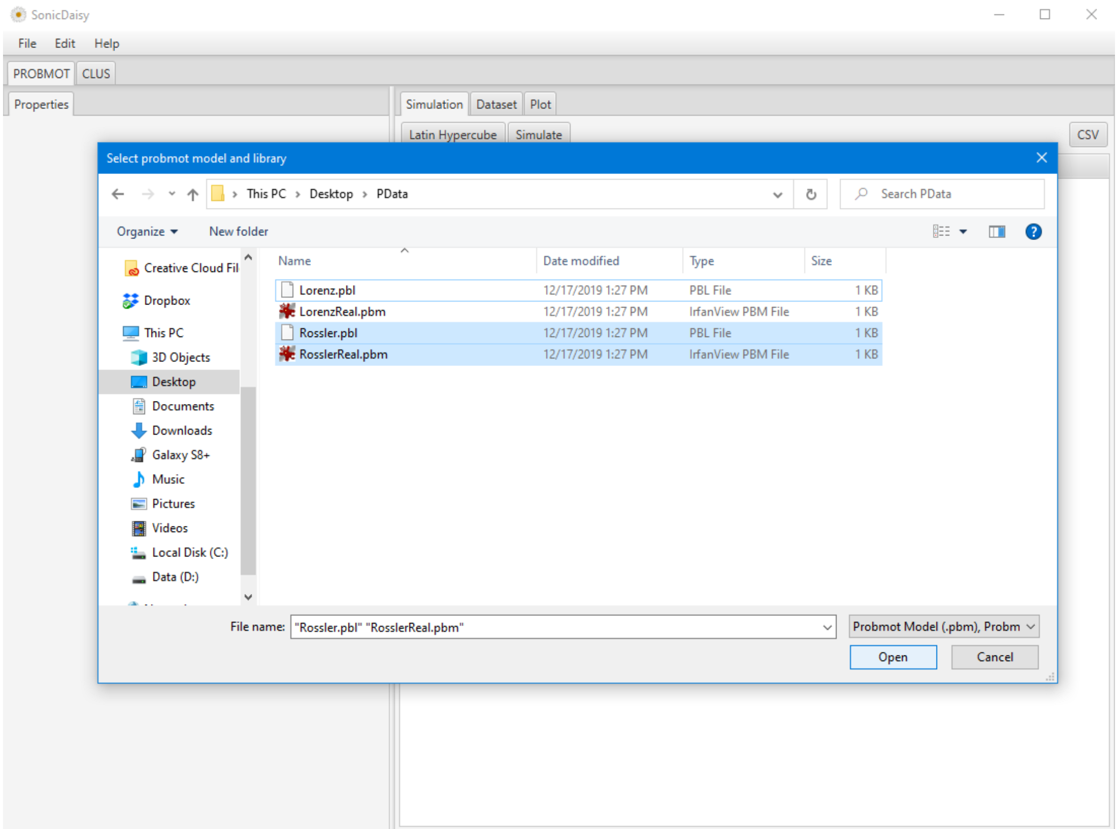

Figure 2: Importing ProBMot model and library to SonicDaisy.

After the ProBMoT files have been imported successfully, in the left part of the main window the properties of the selected ProBMoT model will be shown, see Fig 3. For each property of the given ProBMoT model, we can change its own specific Initial Value and Ranges by double-clicking on the value, changing it and pressing Enter key.

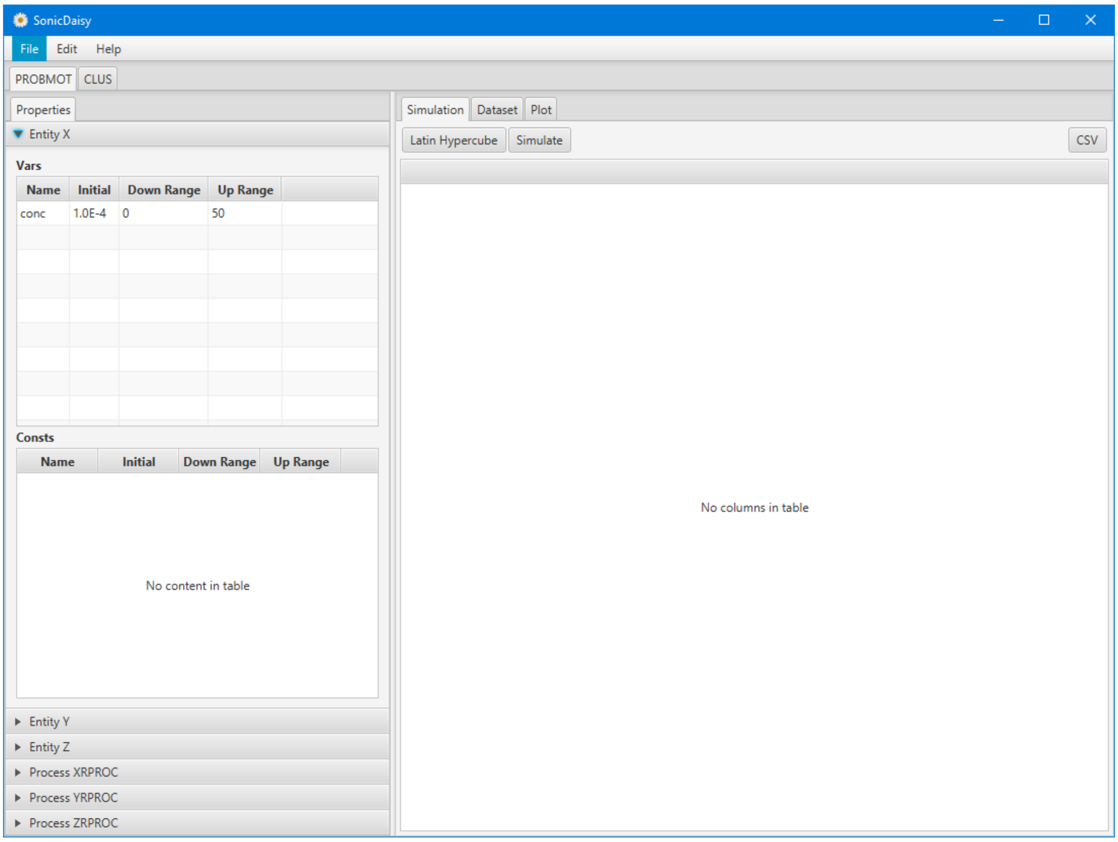

Figure 3: ProBMoT properties shown in SonicDaisy after successful parsing.

The next step is to build a data set by using Latin Hypercube Sampling (more about LHS at section 4) on entities and processes of the selected ProBMoT model. The LHS tool allows the user to run a specific number of the simulations of the given ProBMoT model (see Fig. 4). Also, it allows us to choose between Sensitivity Analysis or Stability Analysis by checking in which entities or processes we want to run LHS (see Fig. 4). If the LHS is disabled then the initial value of the model property is taken.

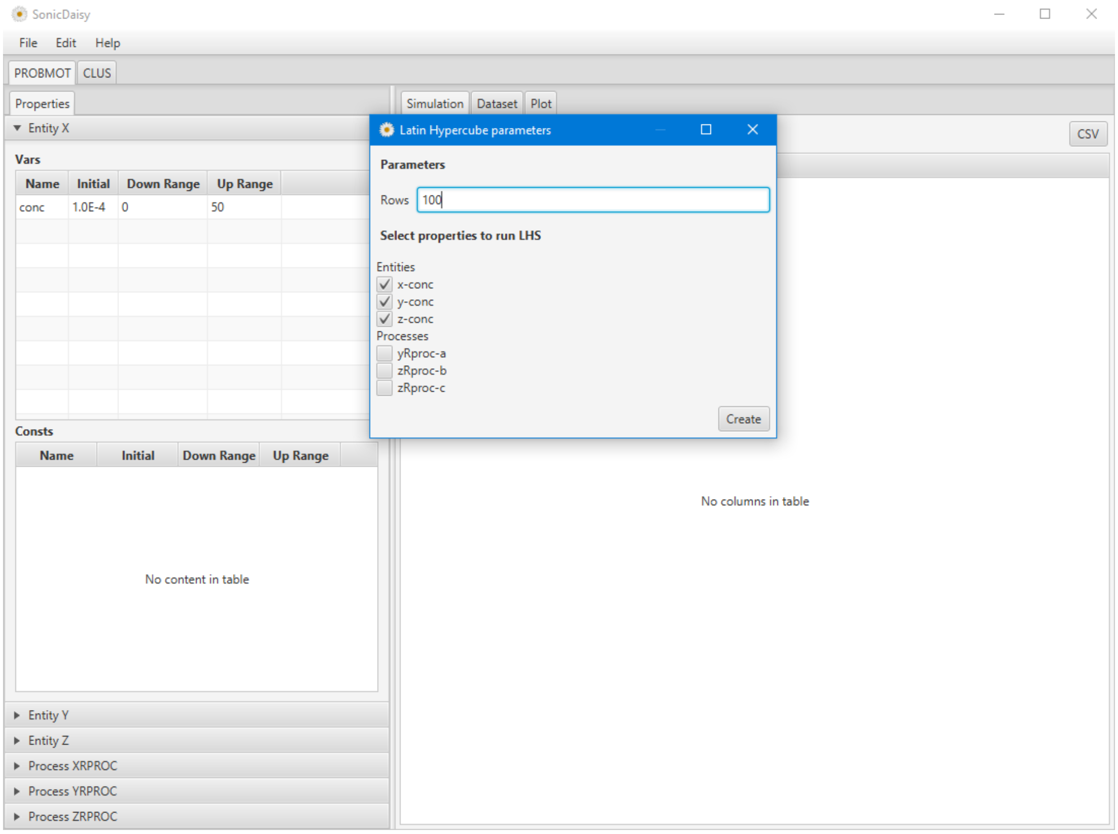

Figure 4: ProBMoT properties shown in SonicDaisy after successful parsing.

After we are set with the *LHS* configuration we proceed to build the first step of the data set by clicking the *Create* button.

The results of the *LHS* are shown in a Table of the main window (see Fig. 5).

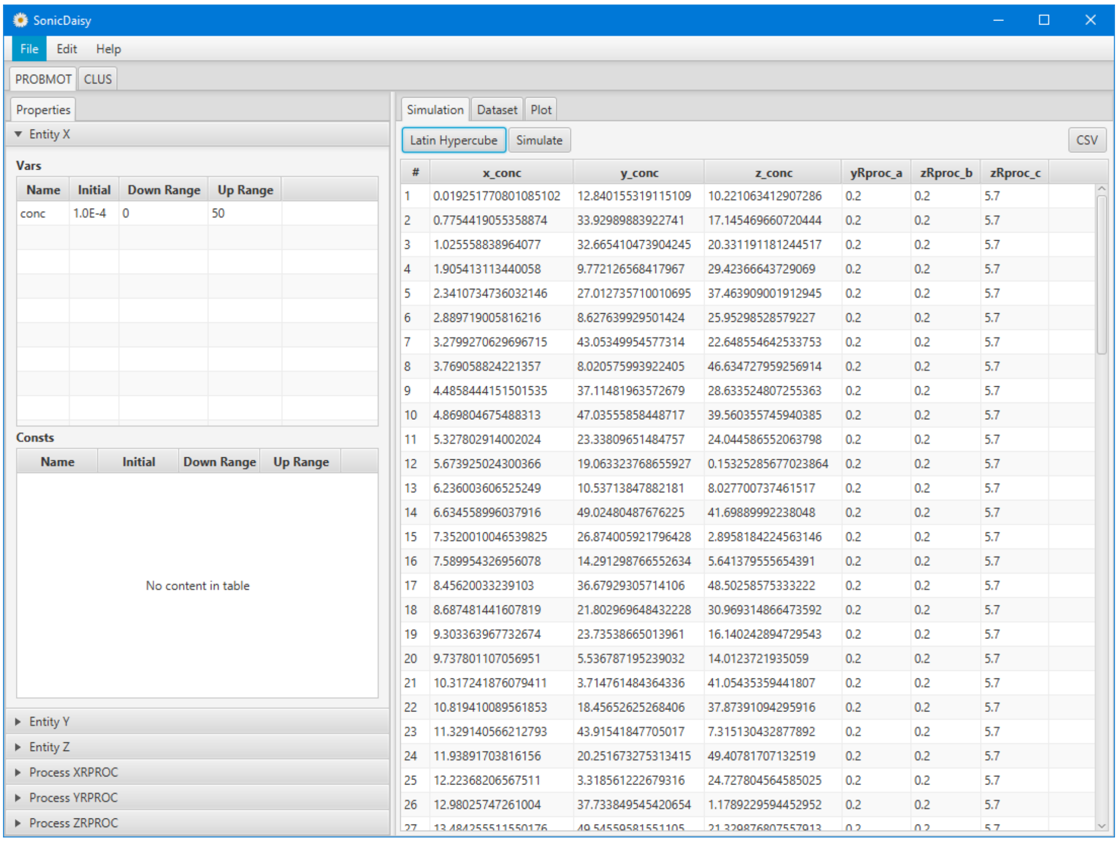

Figure 5: Latin hypercube sampling over the selected ProBMoT properties.

Once we got the input values of the ProBMoT model we can simulate each of it by clicking over the Simulate button just next to the Latin Hypercube button. In the simulation window, the user needs to specify how many time points it wants to run the simulation as well as the step of the time points (see Fig. 6).

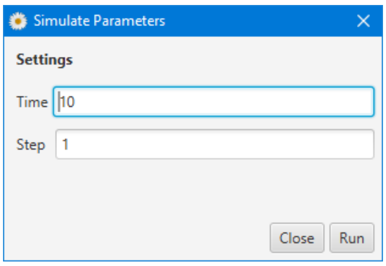

Figure 6: The simulation window.

Once the user is ready it can press the Run button and wait for the simulation to end. After the simulations are finished successfully an alert is shown and a data set is created. This data set is shown under the Tab named *Dataset* under the tab named *raw* since there is no transformation performed on the target values (see Fig. 7). The user can at anytime export locally the created data sets by clicking at the top right button named CSV.

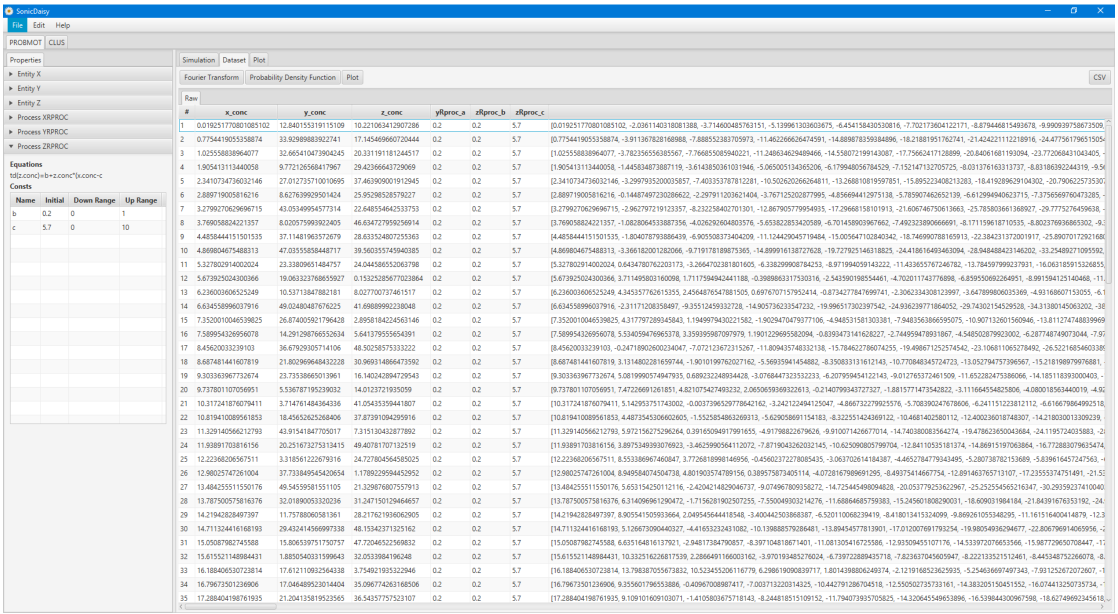

Figure 7: Raw data set shown in the main window under *Dataset* Tab.

After the gained results the user can plot singe, multiple, or all simulations. This is done by holding the *Ctrl* button and clicking over the rows of the Raw data set. After selecting the desired row to plot, the user has to click at the button named Plot (more about charts at 6). The chart results are shown in Fig. 8.

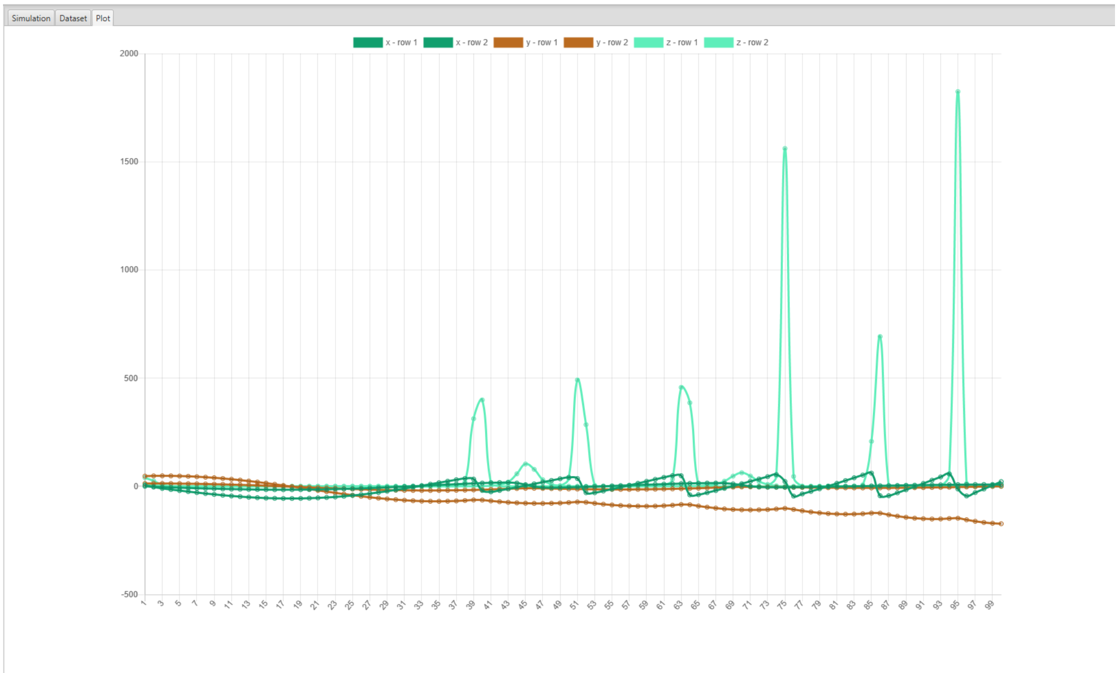

Figure 8: Raw data set shown in the main window under *Dataset* Tab.

SonicDaisy offers at this stage the possibility of transforming the data set target values to Fourier Transform or Probability Density Function. This is achieved by clicking the button named Fourier Transform or Probability Density Function. Once the user clicks the newly transformed data set will be created and shown under the Tab *Dataset* with its name (see Fig. 9 and Fig. 10). To point out again each data set can be exported in CSV format by clicking on the CSV button on the top right part of the main window.

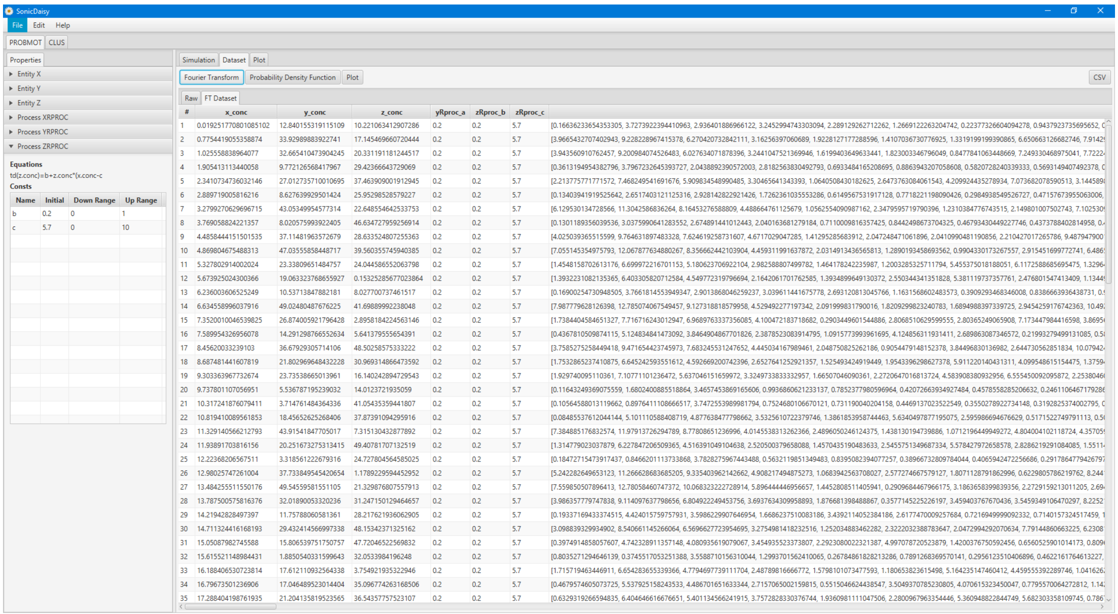

Figure 9: Fourier Transform data set shown in the main window under *Dataset* Tab.

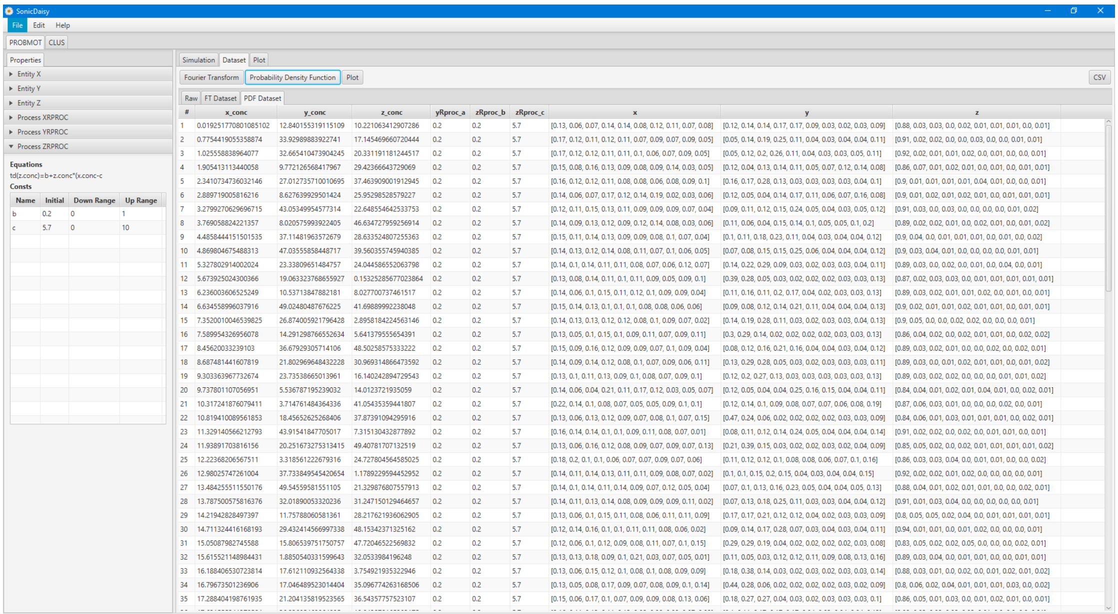

Figure 10: Probability Density Function set shown in the main window under *Dataset* Tab.

Once the user reaches this stage, it can end up with three types of data sets ready to continue to learn from them by using Clus.

In order to use Clus the user needs to select the most top Tab named *Clus*. The *Clus* tab is composed of the left side where the user can see the Clus attributes and data file settings shown in Fig. 11 (more about Clus at 7).

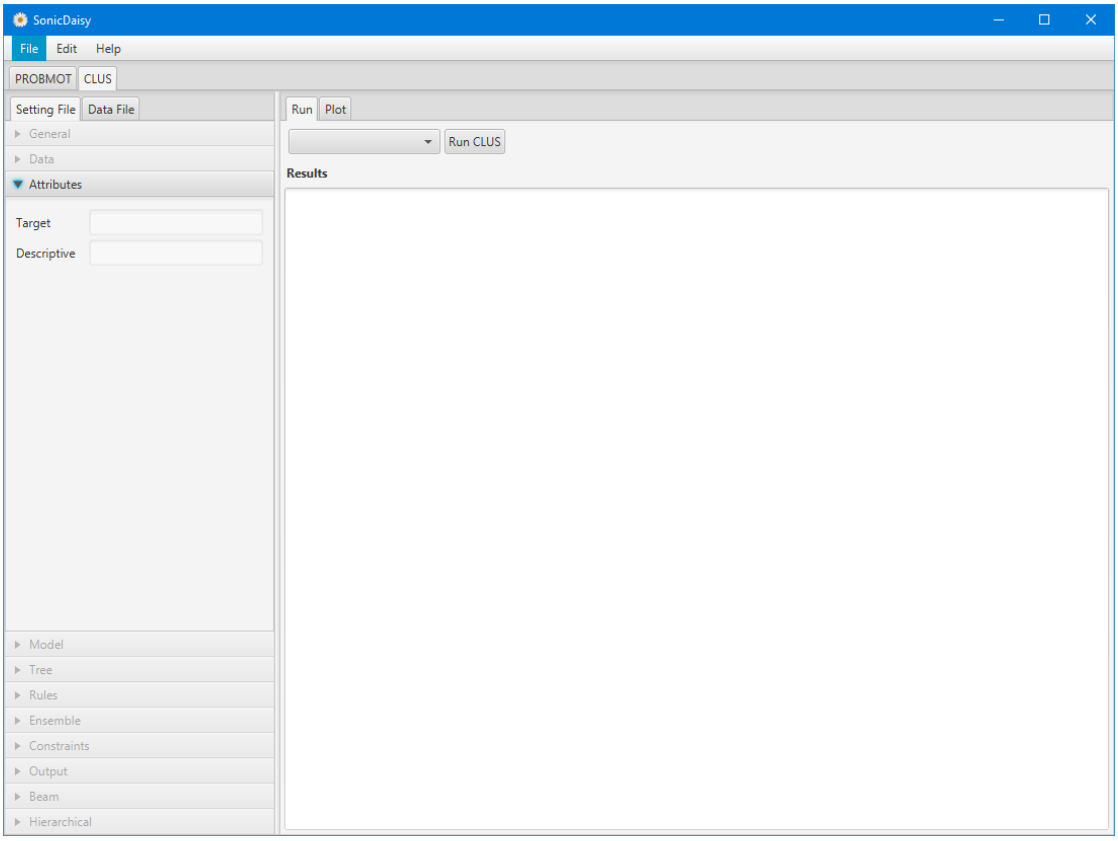

Figure 11: The main window of SonicDaisy - Clus window.

The first step at this stage is to specify in which of available data sets we want to run Clus. This is done by choosing the data set at the drop-down menu on the right side. This drop-down widget will show available data sets as shown in Fig. 12.

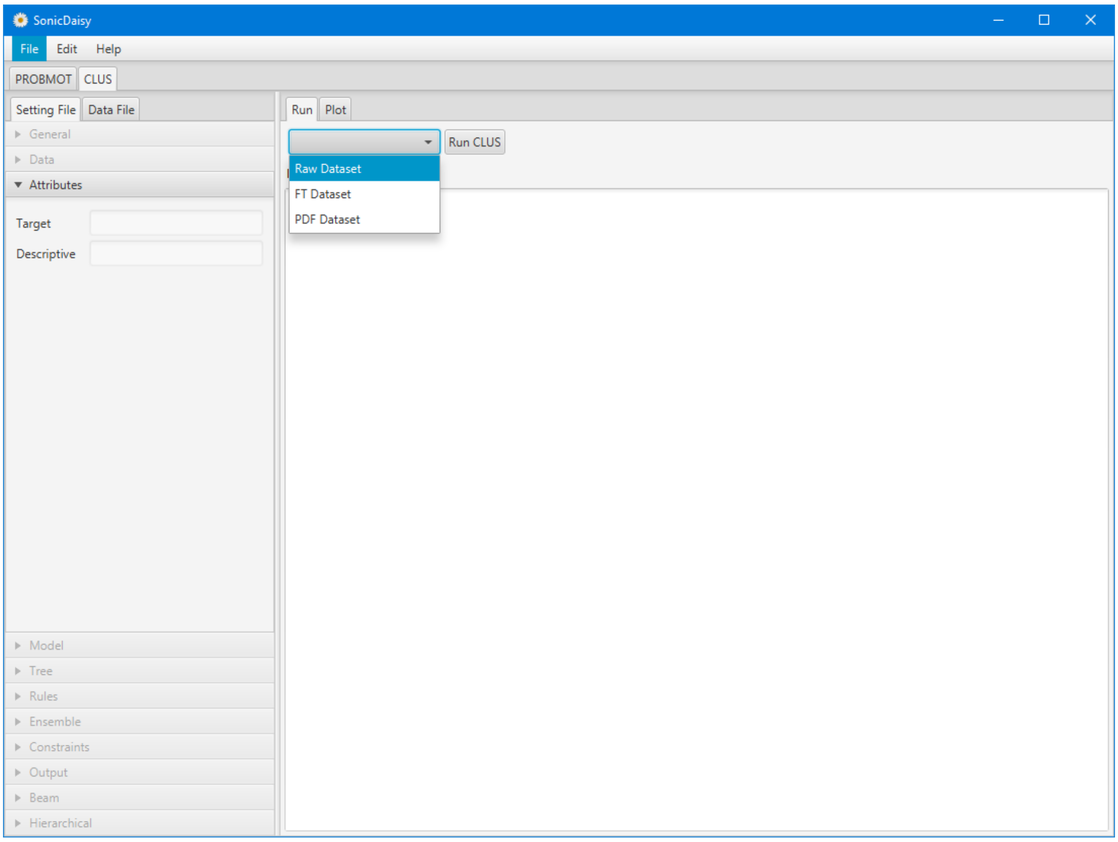

Figure 12: Available data sets to run Clus on.

Once the data set is selected all that is needs to be done is to click the button named Run Clus. After the Clus runs it will show the results as shown in Fig. 13.

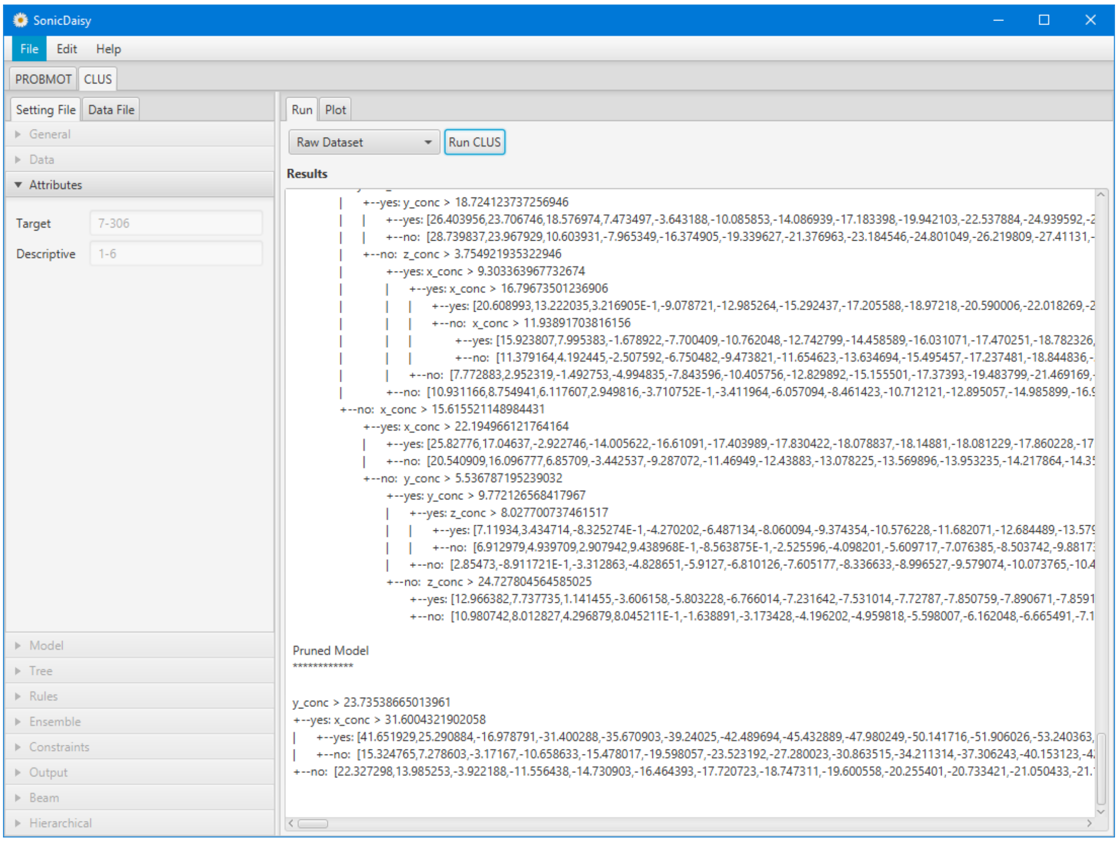

Figure 13: Results of Clus after running on a selected data set.

The same procedure can be done for the rest of the data sets. In order to plot specific leafs of the learned Decision Tree from Clus, the user needs only to copy the results without square brackets (i.e. 22.327298,13.985253,...,19.817768,16.098367,20.731447) and to paste it under the text field labeled *Array* under the tab named *Plot* of the same window. After that all is need is to click the button Plot and the chart will be shown as in Fig. 14.

Figure 14: Visualized results of one Decision Tree Leaf after running on a selected data set.

3 ProBMoT Parser

The ProBMoT parser is a library developed within this project that is used to parse the ProBMoT files like models and libraries.

Above is an example of how to parse a ProBMoT model and to get its own properties in connection with a ProBMoT library.

```java
// Read probmot file with extension .pbm
ModelParser modelP = new ModelParser("RosslerReal.pbm");
// parse probmot model and save it as a Model instance
Model modelRossler = modelP.ParseModel();
// get as list its entitites
List<ModelEntity> modelEntities = modelRossler.getEntities();
// Now read the library that belongs to the correct model
LibraryParser libraryP = new LibraryParser("Rossler.pbl");
// After reading is done parse it
Library libraryRossler = libraryP.ParseLibrary();
// to read all the entites and the entities vars and consts
System.out.println("=== Entities ===");
for (ModelEntity mEntity : modelEntities)
{
    // Get library entity of this model entity
    LibraryEntity e = libraryRossler.findEntity(mEntity.getInherit());
    System.out.println(mEntity.getName() + " " + mEntity.getInherit() + ":" + e.getName());
    // Get model vars
    List<ModelVars> vars = mEntity.getVars();
    for (ModelVars var: vars)
    {
        // try to find a library var for this model var
        LibraryVars libVar = e.findVar(var.getName());
        if(libVar != null)
            System.out.println(libVar.getName() + " " +libVar.getDownRange() + " " + libVar.getUpRange());
        else
            System.out.println("Null");
    }

    // get model consts
    List<ModelConsts> consts = mEntity.getConsts();
    // read all model consts
    for (ModelConsts aConst: consts)
    {
        // find const properties from library
        LibraryConsts libConst = e.findConst(aConst.getName());
        if(libConst != null)
            System.out.println(libConst.getName() + " " +libConst.getDownRange() + " " + libConst.getUpRange());
```

```
        else
            System.out.println("Null");
    }
    System.out.println();
}

// to read all model processes
// first get all processes
List<ModelProcess> modelProcesses = modelRossler.getProcesses();
System.out.println("=== Process ===");
for (ModelProcess mProcess : modelProcesses)
{
    // get process properties from libary
    LibraryProcess p = libraryRossler.findProcess(mProcess.getInherit
    ());
    if(p == null)
        return;
    System.out.println(mProcess.getName() + " " + mProcess.getInherit
    () + ":"+p.getName());
    // get all equations
    System.out.println("EQUATIONS");
    for (String s : p.getEquation().getEquation())
        System.out.println(s);
    // get all consts of the process
    List<ModelConsts> consts = mProcess.getConsts();
    for (ModelConsts aConsts : consts)
    {
        LibraryConsts libConst = p.findConst(aConsts.getName());
        if(libConst != null)
            System.out.println(libConst.getName() + " " +libConst.
    getDownRange() + " " + libConst.getUpRange());
        else
        System.out.println("Null");
    }
    System.out.println();
}
```

4 Latin Hypercube Sampling

Latin Hypercube Sampling is used from a third part library named Commons Math. This library is a packed *.jar* file [4] that contains Latin Hypercube Sampling as is used as shown above.

```
// create an instance of latin hypercube with given m*n size
LatinHypercube lh = new LatinHypercube(m, n);
// this is the random stream base
GenF2w32 stream = new GenF2w32();
// create a random object
Random rd = new Random();
// create an array of a size 25 and fill it with random values
int[] arr = new int[25];
for (int i = 0; i < arr.length; i++)
    arr[i] = rd.nextInt();
// add this random array to the stream
stream.setSeed(arr);
// randomize this stream at the LH object
lh.randomize(stream);
```

5 Fourier Transform and Probability Density Function

Fourier Transform library is built within the scope of this project. The Fourier Transform library can make only the forward transformation of discrete values as shown above.

```
double[] inp = {2.4735765835448222, 1.7549064393333687, ...,
    0.021655999837795613, 0.35453789680909886, 39.31566255465762,
    763.3730046345597};
// create a instance of ClusFFT with a input of double values
ClusFFT mf = new ClusFFT(inp);
// get results in an double array
double[] res = mf.fft();
```

The same library have also the possibility of creating probability density function (PDF) of a given array of values. The PDF is able to be performed in different bin sizes but in the SonicDaisy software is set to bin size 10. The usage of PDF is shown above.

```
double[] inp = {2.4735765835448222, 1.7549064393333687, ...,
    0.021655999837795613, 0.35453789680909886, 39.31566255465762,
    763.3730046345597};
int binSize = 10;
// create a ClusPDF instance with a give input and a specific bin
    size
ClusPDF pdf = new ClusPDF(inp, binSize);
// get results in an double array
double[] res = pdf.pdf();
```

6 Data sets Visualization

The visualization of data sets is done using the web view within the SonicDaisy software. The visualization is done using the Chartjs library which is a JavaScript-based library [3]. In SonicDaisy software, we build a specific *.js* file every time we want to plot something and then we write it to a *HTML* file and then we reload the web view. The code above shows how the *.js* file is built.

```java
private void prepareJS(){
    int tempPos = 0;
    StringBuilder jsOutput = new StringBuilder();
    this.jsOutput.append("var data = { labels: [");
    int labelCounter = this.dataForPlot.get(0).split(",").length;
    for (int i = 1; i <=labelCounter ; i++)
        this.jsOutput.append(i).append(", ");

    this.jsOutput.setLength(this.jsOutput.length() - 2);
    this.jsOutput.append("], datasets: [");

    String colorCode = colorCode();
    int rowPos = 1;
    // dataForPlot is populate with data from SonicDaisy.
    int eachMod = this.dataForPlot.size() / this.colNames.size();
    for (int i = 1; i <= this.dataForPlot.size(); i++) {
        this.jsOutput.append("{").append("label: '").append(this.
    colNames.get(tempPos)).append(" - row ").append(rowPos).append("',
    ");
        this.jsOutput.append("data: ").append(this.dataForPlot.get(i
    -1)).append(", ");
        this.jsOutput.append("borderColor: '").append(colorCode).
    append("', ");
        this.jsOutput.append("fill: false");
        if(i%eachMod == 0)
            tempPos++; colorCode = colorCode();  rowPos = 1;
        else
            rowPos++;
        this.jsOutput.append("},");
    }
    this.jsOutput.setLength(this.jsOutput.length() - 1);
    this.jsOutput.append("]};");
}
// method to generate a random color
private String colorCode(){
    Random obj = new Random();
    int rand_num = obj.nextInt(0xffffff + 1);
    return String.format("#%06x", rand_num);
}
```

7 Clus Builder

The Clus Builder is a library developed within the scope of this project. The goal of this library is to build needed information in order to run Clus such as $.s$ and $.arff$ files. As well as is used to convert data types of type *timeseries* to *numeric*. The Clus Builder library is developed in such way that can be extended to support all the Clus Features.

The above example shows how to build the Setting file $.s$ using blocks. The example is based on the well known $Weather$ data set.

```
// this can have a random seed in constructor
SettingGeneral generalBlock = new SettingGeneral();
// specify data file
SettingData dataBlock = new SettingData("weather.arff");
// specify descriptive and target data
SettingAttributes attributesBlock = new SettingAttributes("1-2","3-4");
// prepare a list of settings
List<Setting> settingBlocks = new ArrayList<>();
// add all blocks to the Setting file
settingBlocks.add(generalBlock);
settingBlocks.add(dataBlock);
settingBlocks.add(attributesBlock);
// build setting file
SettingFile sf = new SettingFile(settingBlocks);
// save this output to a file
sf.showSettingBlocks();
```

The above example shows how to build a simple data file $.arff$ using blocks. This code is based also in the same example $Weather$.

```
// create a realation block and give it a name
ArffRelation r = new ArffRelation("weather");
// create attributes
ArffAttribute outlook = new ArffAttribute("outlook", "{sunny, rainy, overcast}");
ArffAttribute windy = new ArffAttribute("windy", "{yes, no}");
ArffAttribute temperature = new ArffAttribute("temperature", "numeric");
ArffAttribute humidity = new ArffAttribute("humidity", "numeric");
// create datas
ArffData d = new ArffData();
String[] r1 = {"sunny", "no", "34", "50"};
String[] r2 = {"sunny", "no", "30", "55"};
String[] r3 = {"overcast", "no", "20", "70"};
String[] r4 = {"overcast", "yes", "11", "75"};
String[] r5 = {"rainy", "no", "20", "88"};
// remove brackets
d.addRow(Arrays.toString(r1).replace("[", "").replace("]", ""));
```

```
d.addRow(Arrays.toString(r2).replace("[", "").replace("]", ""));
d.addRow(Arrays.toString(r3).replace("[", "").replace("]", ""));
d.addRow(Arrays.toString(r4).replace("[", "").replace("]", ""));
d.addRow(Arrays.toString(r5).replace("[", "").replace("]", ""));
// add them to a list to prepare it for a file
List<ArffElement> arffElements = new ArrayList<>();
arffElements.add(r);
arffElements.add(outlook);
arffElements.add(windy);
arffElements.add(temperature);
arffElements.add(humidity);
arffElements.add(d);
// create a arff file
ArffFile arffFile = new ArffFile(arffElements);
// save this output to a file
arffFile.showArffElements();
```

References

[1] ProBMot, “Proces based modeling tool.” http://probmot.ijs.si/, July 2020.

[2] Clus, “Software for predictive clustering.” http://clus.sourceforge.net/doku.php, July 2020.

[3] Chartjs, “Simple html5 charts using the <canvas> tag.” https://www.chartjs.org/, July 2020.

[4] C. Math, “The apache commons mathematics library.” https://commons.apache.org/proper/commons-math/, July 2020.