A Graphical Tool for Visualizing Bernoulli Stochastics

doi:10.3991/ijet.v4i3.900

Xiaomin Zhai, Elart von Collani
Faculty of Mathematics and Computer Science, Würzburg, Germany

Abstract—**This paper is the continuation of the work of the article "Strategies for Teaching a Novel Approach to Handling Uncertainty Scientifically via Internet". The described novel approach is named Bernoulli Stochastics and one big barrier for introducing it is the prevailing way of deterministic thinking in conjunction with the mathematical language used by Bernoulli Stochastics. To overcome this barrier, an illustrative graphical representation of stochastic concepts and procedures is of crucial importance. In this paper, a flexible and extensible graphical system is described, which was developed to support the understanding of Bernoulli Stochastics by the visualization of uncertainty.**

Index Terms—**Flexible Extensible Graphical System, Mathematical Language, Uncertainty**

I. INTRODUCTION

Uncertainty about the future development represents the main source of the problems of mankind. In order to solve a problem, reliable and accurate predictions are therefore needed. This was identified by Jakob Bernoulli more than 300 years ago and he proposed to develop a science of prediction, which he named Stochastics. This proposal was resumed by Elart von Collani [1, 2, 3, 4], who further developed it to a unified theory of uncertainty about future developments. The aim of this theory is to make reliable and accurate predictions. For disseminating the ideas and results via Internet, the project *Stochastikon*[1] was founded to develop and make available stochastic concepts and procedures for handling the inherent uncertainty about the future development appropriately.

In [5], the main differences between Bernoulli Stochastics and other exact sciences, the origin of the *Stochastikon* framework and the tasks of its subsystems, and especially the specific difficulties in teaching and learning Bernoulli Stochastics are briefly introduced. One of the *Stochastikon* components is a graphical system called *Stochastikon Graphics*, which supports the other components with visualizations of stochastic concepts and procedures.

Being a new branch of science, Bernoulli Stochastics is so far not a part of the traditional education system, thus, there is no classroom teaching and there is no textbook for the dissemination of it. Besides, Bernoulli Stochastics is essentially based on sets, system of sets and relationships between sets. In contrast, traditional science and, hence, traditional education is based on real-valued functions that lead to thinking in points, which is incompatible with Bernoulli Stochastics. In order to overcome these barriers, visualization appears to be the most promising measure, as has been shown in many scholarly papers from the theoretical [6, 7, 8, 9, 10] as well as practical [11, 12, 13] point of view in the case of other mathematical based branches of science. As a matter of fact, visualization of abstract mathematical objects by 'living images' instead of 'frozen images' is looked upon as a milestone in the evolution of mathematical education. Therefore, a proper way for visualizing sets and set functions for the Bernoulli Stochastics teaching is necessary. *Stochastikon Graphics* is not only capable of generating and displaying graphics of sets and set functions easily, but also flexible to be extended to construct different kinds of representing styles demanded by the different *Stochastikon* subsystems.

In the following sections, the strategies and technologies in

- constructing a flexible graphical core, which is independent of data source and extensible for different display forms, and
- accomplishing a multi-purpose graphical system for visualizing a sophisticated mathematics (-involved) discipline, for instance Bernoulli Stochastics,

are briefly discussed and the implementations of these ideas are demonstrated in Java programming language.

II. CONSTRUCTING A GRAPHICAL SYSTEM

A. Plotting Objects

Points, lines, curves, shapes, polygons, labels and axes are the basic plotting objects for either two- or three-dimensional graphs produced by *Stochastikon Graphics*. These plotting objects are briefly described as follows:

- 'Point': a point is determined by a set of coordinates (x, y) or (x, y, z). 'Point' is the most basic plotting object and a function is a description of points with special functional relations.
- 'LineStyle' determines the color, line width, point shape (star, round, cross...), point size of a plotting object.
- 'FillStyle' determines the filling direction (to X-, Y- or Z-axis), filling or outlining style (with points, lines, dots, a combination of lines and points, etc) of a plotting object.
- 'StyledPoint' is a 'Point' with an assigned plotting style.
- 'MarkedPoint' is a 'Point', from which a vertical line to each axis is drawn with a specific plotting style. 'MarkedPoint' marks out a given position.

[1] http://www.stochastikon.com/

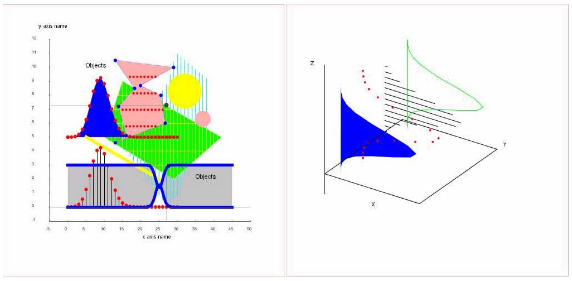


Figure 1. The basic plotting objects

- ‘Label’ writes characters or numbers in a specified position.
- ‘Line’ draws a line between two points with an assigned line style.
- ‘Curve’, ‘Shape’ and ‘Polygon’ are graphed according to one or two given functions. ‘Polygon’ can also be drawn according to a series of given points.
- ‘Axes’ fix the dimension character of a graph. Each axis has a name, a type (continuous or discrete) and bounds.

A three-dimensional plotting object is constructed by adding one dimension (by a value or a series of values) to an existing two-dimensional plotting object.

Fig. 1 illustrates the plotting objects. On the right part of Fig. 1, different FillStyles are displayed. They are, from left to right: full fill blue, no fill red point-outline, line fill black, no fill green line-outline. The polygons are graphed based on the function $x = f(z)$ plotted on x-z plane, while different polygons are differentiated by their y values.

Basic methods of each plotting object are listed in Table I.

There are corresponding methods for the *Z*-axis in the case of a three-dimensional plotting object. With the methods listed in Table I, plotting objects can be incorporated and plotted.

‘PlotObjectSet’ is a special kind of plotting object. It has an additional method ‘add(IPlotObject Object)’ to combine several plotting objects to a bigger one, i.e. a plotting object set. In the left part of Fig. 2, there is a fish. The fish is a PlotObjectSet, which contains Curves, StylePoints, Polygons and Lines.

A PlotObjectSet can be treated as an element-plotting-object within a larger plotting object set. The right part of Fig. 2 gives an example. The PlotObjectSet ‘fish’ together with some other plotting objects compose a new PlotObjectSet, which contains four different kinds of fishes.

TABLE I.
BASIC METHODS OF THE PLOTTING OBJECTS

void	addToPlotData(PlotData data)	Add the plotting object into a PlotData.

IInterval	getPreferredXRange()
	getPreferredYRange()
		Return an interval, which indicates the preferred plotting range of the plotting object on the *X*- or *Y*-axis.

IInterval	getMaximumXRange()
	getMaximumYRange()
		Return an interval, which indicates the maximum plotting range of the plotting object on the *X*- or *Y*-axis.

int	getPreferredXType()
	getPreferredYType()
		Return an integer, which indicates the type of *X*- or *Y*-axis of the plotting object.

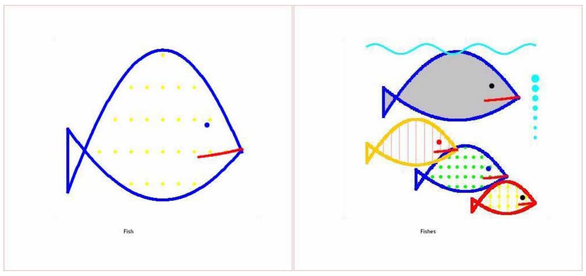


Figure 2. Examples of PlotObjectSet: a fish and several fishes

B. *Plotting Data*

The plotting data (‘PlotData’) stores, transfers and offers plotting objects for drawing figures. The most important methods for plotting data are listed in Table II.

There are two ways to ‘add’ the plotting objects to a three-dimensional plotting data:

- If the plotting object is three-dimensional, then the method is analogous to ‘addMarkedPoint(StyledPoint3d point)’.
- If the plotting object is two-dimensional, then another dimension must be added and the method is analogous to ‘addMarkedPointXY(StyledPoint2d point, double z)’.

By means of special algorithms, the plotting objects are drawn especially for the ‘Shape’ and the ‘Polygon’. For example, when ‘filling’ a polygon, it is necessary to determine every pair of symmetrical points in the determined plotting direction. To this end, counting algorithms under side conditions, for instance with regard to concavity or convexity, are required.

C. *Generating Graphs*

Each PlotData contains concrete plotting objects and/or PlotObjectSets for composing a complete picture. There are various ways to generate graphs from PlotData, by programming or by means of a graphical toolkit. Below two examples are given that are implemented in *Stochastikon Graphics*:

- Graphs may be generated with a graphical software package, such as Gnuplot [14].

The PlotData are transformed by a class named ‘GnuplotFile(IplotData plotData, int outputFiletype, String path, String size, int tics)’ into a *.gnu file. The main function of this class is to change the plotting objects contained in the PlotData into the corresponding Gnuplot drawing commands and then store these commands in a drawable *.gnu file. The Gnuplot software creates a graph based on the *.gnu file and saves the graph as *.eps, or *.gif or *.tex file. Fig. 3 depicts the working flow.

- Graphs may be generated by programming, such as Java programs [15].

By means of several classes, the plotting objects stored in the PlotData are transformed into standard ‘Java.awt.Graphics2D’ objects. With these classes, plotting objects are converted into a Java image, which can be

TABLE I.
BASIC METHODS OF THE PLOTDATA

int	getCurveLineStyle(int index)
int	getCurvePlotStyle(int index)
LineStyle[]	getLineStyles()
int	getPolygonFillStyle(int index)
int	getPolygonLineStyle(int index)
int	getShapeFillStyle(int index)
int	getShapeLineStyle(int index)
Axe	getXAxe()
Axe	getYAxe()
Axe	getZAxe() <for three-dimensional Plotting Objects>
int	numberOfCurves()
int	numberOfPolygons()
int	numberOfShapes()
Point	getCurvePoints(int index)
Label	getLabels()
Line	getLines()
StyledPoint	getMarkedPoints()
StyledPoint	getPoints()
Point	getPolygonPoints(int index)
Point	getShapePoints(int index)
void	addLabel(Label label)
void	addLine(Line arrow)
void	addLineStyle(LineStyle style)
void	addMarkedPoint(StyledPoint point)
void	addPoint(StyledPoint point)
void	addCurve(Point [] points, int lineStyle, int plotStyle)
void	addShape(Point [] points, int lineStyle, int fillStyle)
void	addPolygon(Point [] points, int lineStyle, int fillStyle)

displayed on the screen, saved as *.jpeg or *.png file or stored in form of a Java image stream.

A graphical toolkit like Gnuplot has certain functions, which support the plotting of graphs directly. Therefore, to generate images and graphical files from the PlotData by Java programs is comparably more difficult since classes must be provided, for example, for plotting tics and tic labels of axes, plotting three-dimensional figures, etc. However, the result formats are more flexible and have broader applicability.

D. *Presenting Graphs*

The generated images and graphical files can be displayed in different styles for different purposes: they can be presented alone or inserted into other documents, demonstrated in an interactive dynamic way or just as a static graph, dynamically displayed in a stand-alone computer or via Internet, etc. The different layout forms can also be realized in different ways. In the following section, three graphical systems accomplished by *Stochastikon Graphics* are introduced in detail as examples.

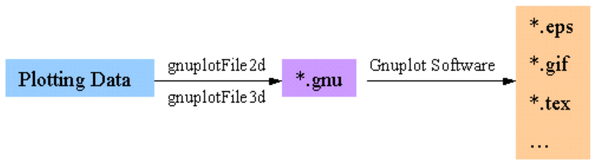


Figure 3. Generating a graphical file with Gnuplot

III. STOCHASTIKON GRAPHICS

As introduced in [5], *Stochastikon* is a comprehensive information system on Bernoulli Stochastics that consists of several subsystems. Each subsystem takes a clear and definite task within the whole framework. There is the subsystem *Magister* for E-Learning, the subsystem *Calculator* for computing, the subsystem *Graphics* for plotting, etc. The subsystem *Calculator* is the calculating engine for *Graphics* and *Graphics* has to satisfy all kinds of graphical requests from the other subsystems. The construction design of the above mentioned graphical system provides the possibility for unrestricted types of data sources and various kinds of approaches to implement and extend it as a multifunctional graph provider for a mathematics (-involved) discipline. *Stochastikon Graphics* is a good example to demonstrate these features.

A. *Stochastic Objects*

Stochastic objects are PlotObjectSets composed for visualizing the concepts and procedures of Bernoulli Stochastics. Table III lists the stochastic objects (for details see [1, 2, 3, 4]) and the corresponding plotting objects used to compose them.

In *Stochastikon Graphics*, a realized stochastic object is based on the numerical results obtained by the *Stochastikon Calculator*. The following two classes are of particular importance for this process:

- FunctionParameter1(IFunction f) that handles the cases, when the result of the parameter function *f* is a single value.
- FunctionParameter2(IFunction f) that handles the cases, when the output of the parameter function *f* is a set of values given by an interval.

The parameter function *f* is the result of *Calculator*. Based on the properties of the parameter (continuous or discrete, monotonic, uni-modal, probability density function, etc), these two classes calculate and decide the plotting bounds and the plotting types for representing the stochastic objects.

The procedures of Bernoulli Stochastics have an extremely high mathematical and numerical complexity and represent problems of new and so far unsolved nature. Therefore, unlike general graphical systems that have been implemented in the well-established mathematical teaching areas such as algebra, geometry, statistics and probability theory, *Graphics* has to consider new aspects in order to compose and display the stochastic objects appropriately. For instances,

- the default setting should display the most meaningful part of the stochastic object and, at the same time, the user should be able to select the desired part manually ;
- the tic length and the displayed tic labels for different scales of magnitude within the two- and three-

dimensional graphs must be balanced. For example, the tic length must ensure that there is no overlap in case of long tic labels and the last digit of a tic label must be 2, 5 or 0 (0 is selected as the last digit only in case the scale of the magnitude is larger than 1).

Fig. 4 displays the stochastic object 'Property Function' of the binomial distribution. On the left hand side, there are graphs of several two-dimensional probability functions for the specified parametric values. In the three-dimensional picture on the right-hand side, the blue curve shows a probability mass function of the binomial distribution. The three-dimensional background (the shadow) is made up of probability mass functions of the binomial distribution, which have the same success probability *p* as the blue curve but have different sample size *n*.

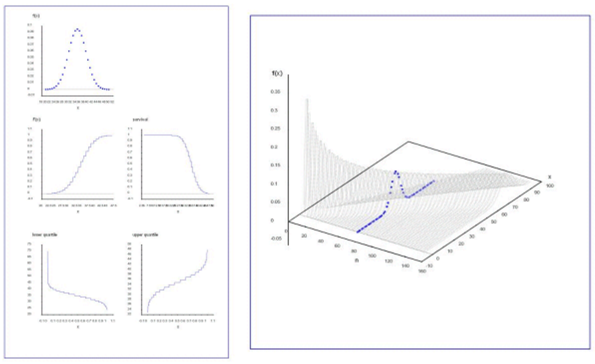


Figure 4. The stochastic object 'Property Function' of the binomial distribution

B. Stochastic Graph Systems

Stochastic objects are stored and transferred by PlotData to implement different kinds of graph systems for visualizing Bernoulli Stochastics in different *Stochastikon* subsystems.

1) Report System

On request, the *Stochastikon* subsystem *Calculator* issues a complete scientific report containing the proposed problem with the given parameter values and the calculated numerical results. In both of these parts of the report, graphical representations show the corresponding stochastic concepts, procedures and results.

The *Calculator* sends both the user requirements and the calculated results to the *Graphics* and asks for some specified illustrations. *Graphics* takes the results and generates the graphs with the help of the Gnuplot software. These graphs are given back to the *Calculator* and inserted into the proper places for completing the final report. The Stochastic Graph System, which supports the creation of the illustrations for the stochastic reports, is called the 'Report System'.

TABLE II.
STOCHASTIC OBJECTS AND THE CORRESPONDING PLOTTING OBJECTS

IV. Stochastic Object		V. Plotting Object

Beta Measurement Space	Lower Bound	Shape2d12(IFunction2dParameter param, LineStyle lineStyle, int fillStyle)
		MarkedPoint2d2(Point2d p, LineStyle style)
		Line2d2(double x1, double y1, double x2, double y2, LineStyle style)
		StyledPoint2d2(Point2d p, LineStyle style)

	Upper Bound	

	Minimum	Shape2d22(IShape2dParameter param, LineStyle lineStyle, int fillStyle)
		MarkedPoint2d2(Point2d p, LineStyle style)
		Line2d2(double x1, double y1, double x2, double y2, LineStyle style)
		StyledPoint2d2(Point2d p, LineStyle style)

		Shape2d12(IFunction2dParameter param, LineStyle lineStyle, int fillStyle)
		MarkedPoint2d2(Point2d p, LineStyle style)
		Line2d2(double x1, double y1, double x2, double y2, LineStyle style)
		StyledPoint2d2(Point2d p, LineStyle style)

Beta Uncertainty & Prediction Space	Lower Bound	Shape2d1(IFunction2dParameter param, int lineStyleIndex, int fillStyle) StyledPoint2d(Point2d p, int style)
	Upper Bound	
	Minimum	
Beta Classification	2 Alternatives	Shape2d1(IFunction2dParameter param, int lineStyleIndex, int fillStyle) Line2d2(double x1, double y1, double x2, double y2, LineStyle style) StyledPoint2d(Point2d p, int style)
	3 Alternatives	
Beta Exclusion		Shape2d1(IFunction2dParameter param, int lineStyleIndex, int fillStyle) Line2d2(double x1, double y1, double x2, double y2, LineStyle style) StyledPoint2d(Point2d p, int style)
Ignorance Space		Polygon2d1(IFunction2dParameter param, LineStyle lineStyle, int fillStyle)
Uncertainty Space		Polygon2d1(IFunction2dParameter param, LineStyle lineStyle, int fillStyle)
Property Function		Curve2d(IFunction2dParameter param, LineStyle lineStyle) Curve2d(IFunction2dParameter param, int lineStyleIndex)

Probability of an Event		Curve2d(IFunction2dParameter param, LineStyle lineStyle)
		Shape2d12(IFunction2dParameter param, LineStyle lineStyle, int fillStyle)

The Report System can be used in a stand-alone computer or via Internet. Users can download or print the final PDF reports generated automatically by the *Stochastikon Calculator*.

1) Stand-Alone-Computer Graph System

When installing the *Stochastikon* system in a local computer, users can use the 'Stand-Alone-Computer Graph System'. It is an interactive dynamic graphical system that supports the understanding and application of mathematics (-involved) knowledge.

The Stand-Alone-Computer Graph System is a multi-lever system. In Fig. 5, the structure of the Stand-Alone-Computer Graph System is depicted.

The most important classes of the Stand-Alone-Computer Graph System are called 'JavaP' and 'P_Component'.

- JavaP is an abstract class used for generating a graph image in a printable container according to different parameter values. It is one of the basic classes of the Stand-Alone-Computer Graph System. All classes, which are extended from this class, are used to settle java swing objects for users in order to input the parameter values.
- P_Component is another basic abstract class of the Stand-Alone-Computer Graph System. All classes extended from this class are used to create graphs according to the parameter values transferred from the corresponding subclass of JavaP.

In the Stand-Alone-Computer Graph System, a user specifies the 'Distribution' and the 'Stochastic Object' at first, and then sets the necessary parameter values and some other choices concerning the stochastic object by slide bars or multiple choices. Along with the change of the parameter values by the user, the graph system plots simultaneously the corresponding figures on the graph panel based on the numerical results from the *Calculator*. Users can detect the graphical change-track following the change of parameter values, and they can also print the graphs whenever they want.

2) Web Graph System

The web application of the stochastic graph system is called 'Web Graph System', which does not need the installation of any specific software in the local computer. It is also an interactive dynamic graphical system. One application of the Web Graph System is the Graphical Laboratory within the E-Learning System *Stochastikon Magister* for practicing stochastic concepts and procedures [5].

After selecting links for the 'Distribution' and the 'Stochastic Object' on the web page, an applet will run and wait for the input parameters and some other choices regarding the selected stochastic object. By clicking the 'ShowGraph' button, the user request is identified and sent to the web server. The resulting graphs are displayed on the applet panel. The operating process of the Web Graph System is almost the same as the Stand-Alone-Computer Graph System, except for the additional confirming button 'ShowGraph'.

The Web Graph System is also a multi-lever system like the Stand-Alone-Computer System. Each of the web pages lists a series of links to *.html files for a specific distribution. Each html file starts a graphical applet for manipulating a stochastic object.

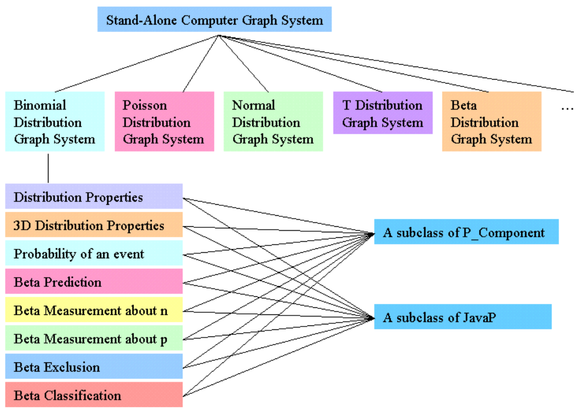

Stand-Alone Computer Graph System
Binomial Distribution Graph System
Poisson Distribution Graph System
Normal Distribution Graph System
T Distribution Graph System
Beta Distribution Graph System
...
Distribution Properties
3D Distribution Properties
Probability of an event
Beta Prediction
Beta Measurement about n
Beta Measurement about p
Beta Exclusion
Beta Classification
A subclass of P_Component
A subclass of JavaP

Figure 5. Structure of the Stand-Alone-Computer Graph System

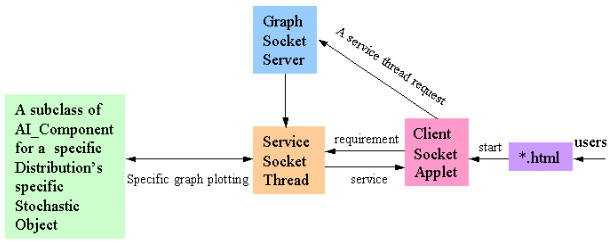


Figure 6. Workflow of the Web Graph System

The workflow of the Web Graph System is illustrated in Fig. 6.

For accomplishing its tasks, the Web Graph System uses Sockets that appeared to be the only way to implement interactive dynamic graphs online for the *Stochastikon* system. This solution is another feature, which distinguishes *Graphics* from general online interactive dynamic graphical systems as used, for example, in didactics of mathematics. Usually the graphical applet running on the web contains the complete calculating and plotting part or stores all the related programs in a jar file. But as for the *Stochastikon* system, the sizes of the *Calculator* and the *Graphics* are too large and in future versions they will be a multitude of the present size. Therefore, it is impossible to zip them into a jar file and run the whole program by applets. The only way is to let some small applets run on the web part, while the *Calculator* and the main part of the *Graphics* run on the server. In fact, this operating mode is very efficient with a short download time on the user side.

The most basic classes for the Web Graph System are called 'AI_Component' and 'GraphServer'.

- AI_Component is an abstract class containing the common methods and variables for generating a graph in a 'BufferedImage' in accordance with the given parameter values. All classes extended from this class are used for drawing graphs on the basis of the parameter values transferred from the corresponding parameter-acceptance Client Socket applet. The result 'BufferedImage' can be saved as a jpeg file or in form of an image stream.
- GraphServer is a Server Socket providing services according to the requests from different "clients". When accepting a demand from a specific Client Socket, it will start a corresponding service Socket Thread. Each Client Socket is a Swing Applet that accepts the user input and sends the parameter values to the Socket Thread. The service Socket Thread then

calls a specific subclass of AI_Component to generate a graph according to the parameter values delivered from the Client Socket. While the graph is accomplished and saved as a *.jpeg file, the Client Socket is informed about the name of the graph. The Client Socket applet will then take the graph from a specific place according to the name and display the graph on its web panel. This running mode enhances the stability of the online graphical representing system. The by-products are the *.jpeg files, which can later be checked and saved.

The Web Graph System allows users to run several Client Socket applets (the graph windows) simultaneously and thus learn by comparison.

The Stand-Alone-Computer Graph System and Web Graph System are both interactive dynamic graphical systems using Java images to store, convey and display graphs. The differences between the web system and the stand-alone-computer system are as follows:

- The Stand-Alone-Computer Graph System shows the generated graph images directly. Therefore, any change of the parameter values yields simultaneously the changed graph. With this real time pattern, it is easier to observe the changing effect of the parameter values.
- The Web Graph System uses applets to show graphs on the web. The created images are stored as *.jpeg files. Although, the Web Graph System can be implemented in the same changing mode as the Stand-Alone-Computer Graph System, the process is not that stable. As can be seen from the workflow of the Web Graph System depicted in Fig. 6, the necessity of using Sockets leads to extensive communications between the Client Socket applet and the service Socket Thread via Internet. Moreover, creating and saving *.jpeg files is time-consuming. Therefore, if the system has to generate a *.jpeg file for every tiny change of the input values along with the sliding of the slider bar, the server would not be able to accomplish the huge workload. Thus, when the Client Socket gets the name of the required graph file, the server may still have not saved the file into the assigned place. An acknowledgement by means of the 'ShowGraph' button is asked for whenever the user has set the values of the input parameters and wants to see the resulting graph.

3) Illustrations of the Three Stochastic Graph Systems

In this section, the characteristic layouts of the three stochastic graph systems implemented by *Stochastikon Graphics* are presented.

Fig. 7 shows the third page of a stochastic report created by the *Stochastikon Calculator*. It contains two illustrations generated by the Report System. These two graphs show the uncertainty space (light blue), the prediction (blue), and the probability of the prediction (the blue curve below) (for details see [1, 2, 3, 4]).

Fig. 8 is a typical layout of an implemented Stand-Alone-Computer Graph System. Users can open more than one inner window simultaneously for different distributions and different stochastic objects.

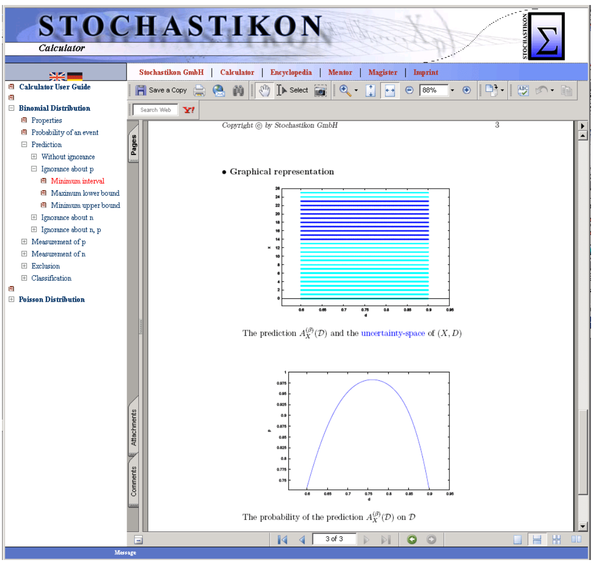


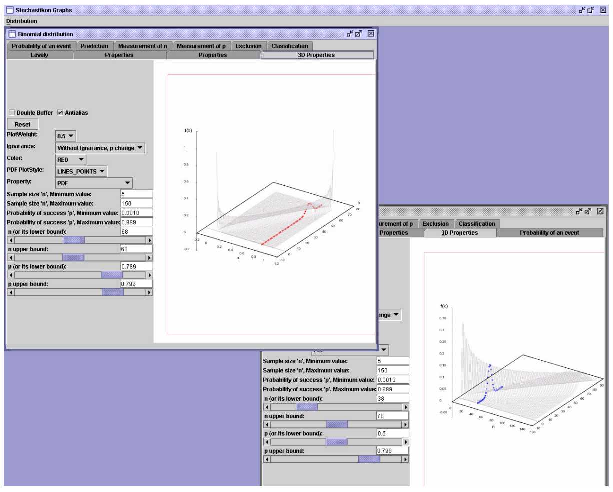
Figure 7. The Report System provides illustrations for the final reports of the *Stochastikon Calculator*

Fig. 9 shows the user interface of a stochastic object (an exclusion procedure) within the Graphical Laboratory of *Stochastikon Magister*. The Graphical Laboratory is an application of the Web Graph System.

VI. CONCLUDING REMARKS

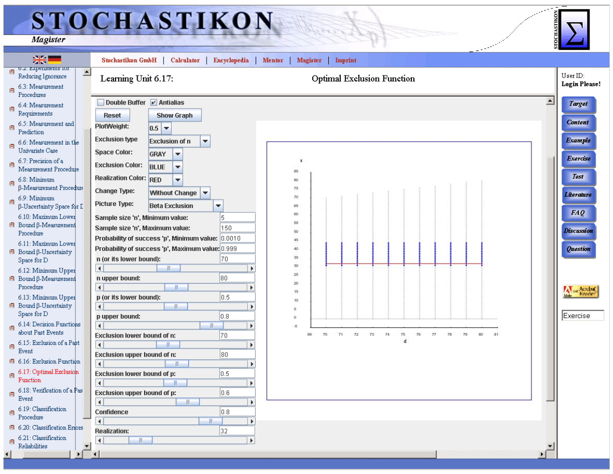
Bernoulli Stochastics and the *Stochastikon* project can revolutionize the handling of uncertainty, which is crucial for any decision making process. The currently described *Stochastikon Graphics* shall contribute to the dissemination of Bernoulli Stochastics and, therefore, is a critical part of the entire project. Feedbacks from users either of *Calculator* or of *Magister* are fully positive towards the visualization of Bernoulli Stochastics, which report a better understanding, a quicker acceptance, a longer memory and an easier operation to the new and sophisticated concepts and procedures, the relationships between stochastic objects and their properties. The user reactions, are no surprise, in consistent with a lot of former conclusions in this research area. Since the first version of *Magister* became available on-line, it was used for the education and training the candidates of the teaching profession majoring in mathematics. Solely based on *Magister* E-Learning system, they took part in an examination in Bernoulli Stochastics instead of the otherwise required examination in statistics. A preliminary comparison shows that the learning effects in Bernoulli Stochastics is significantly better than those obtained in statistics, which is to a large extent due to the graphical representation of models and methods. Currently, the results are analyzed and the details will be published in a separate paper. Along with the future advancements of the *Stochastikon* system, *Stochastikon Graphics* will be further developed and will thus allow more powerful applications.

The core of the graphical system is concise and data source independent, therefore graphs can be based on assigned values (see Fig. 1), on simple functions (see Fig. 2) or on a huge complicated calculating engine like *Calculator* (see Fig. 7, Fig. 8 and Fig. 9). The core is also open for implementing approaches, with which graphs can be pro-



PlotWeight: 0.5
Ignorance: Without Ignorance, p change
Color: RED
PDF PlotStyle: LINES_POINTS
Property: PDF
Sample size 'n', Minimum value: 5
Sample size 'n', Maximum value: 150
Probability of success 'p', Minimum value: 0.0010
Probability of success 'p', Maximum value: 0.999
n (or its lower bound): 68
n upper bound: 68
p (or its lower bound): 0.789
p upper bound: 0.799

Figure 8. Layout of an implementation of the Stand-Alone-Computer Graph System


STOCHASTIKON
Magister
Stochastikon GmbH | Calculator | Encyclopedia | Mentor | Magister | Imprint
6.2: Experiments for Reducing Ignorance
6.3: Measurement Procedures
6.4: Measurement Requirements
6.5: Measurement and Prediction
6.6: Measurement in the Univariate Case
6.7: Precision of a Measurement Procedure
6.8: Minimum β-Measurement Procedure
6.9: Minimum β-Uncertainty Space for D
6.10: Maximum Lower Bound β-Measurement Procedure
6.11: Maximum Lower Bound β-Uncertainty Space for D
6.12: Minimum Upper Bound β-Measurement Procedure
6.13: Minimum Upper Bound β-Uncertainty Space for D
6.14: Decision Functions about Past Events
6.15: Exclusion of a Past Event
6.16: Exclusion Function
6.17: Optimal Exclusion Function
6.18: Verification of a Past Event
6.19: Classification Procedure
6.20: Classification Errors
6.21: Classification Reliabilities
Learning Unit 6.17:
Optimal Exclusion Function
Double Buffer
Antialias
Reset
Show Graph
User ID:
Login Please!
Target
Content
Example
Exercise
Test
Literature
FAQ
Discussion
Question
Exercise

PlotWeight: 0.5
Exclusion type: Exclusion of n
Space Color: GRAY
Exclusion Color: BLUE
Realization Color: RED
Change Type: Without Change
Picture Type: Beta Exclusion
Sample size 'n', Minimum value: 5
Sample size 'n', Maximum value: 150
Probability of success 'p', Minimum value: 0.0010
Probability of success 'p', Maximum value: 0.999
n (or its lower bound): 70
n upper bound: 80
p (or its lower bound): 0.5
p upper bound: 0.8
Exclusion lower bound of n: 70
Exclusion upper bound of n: 80
Exclusion lower bound of p: 0.5
Exclusion upper bound of p: 0.6
Confidence: 0.8
Realization: 32

Figure 9. An application of the Web Graph System, the Graphical Laboratory, for *Stochastikon Magister*

duced in different way and can be demonstrated for different purposes under different running environments. These characteristics make the finalized system to be multi-applicable and serve for both didactic and professional graphical representations. Besides, the described technique over here for accomplishing the Web Graph System solves the bottleneck of realizing an online interactive dynamic visualization of procedures that can be calculated only by extremely complicated mathematical engines.

Actually, visualization is a key for understanding mathematical sciences. Although *Graphics* has been developed in the framework of the *Stochastikon* system, the strategies and technologies applied for designing and implementing it can be taken as a model for constructing a flexible, extensible and reusable graphical system to visualize any mathematics (-involved) discipline.

REFERENCES

[1] E. von Collani, "Theoretical Stochastics" in *Defining the Science of Stochastics*, E. von Collani, Ed. Lemgo: Heldermann Verlag, 2004, pp. 147-174.

[2] E. von Collani, "Empirical Stochastics" in *Defining the Science of Stochastics*, E. von Collani, Ed. Lemgo: Heldermann Verlag, 2004, pp. 175-213.

[3] E. von Collani and X. Zhai, *Stochastics*, Beijing: Beijing Publisher Company Group, 2005.

[4] E. von Collani, "Defining and Modeling Uncertainty", *Journal of Uncertain Systems*, 2008, vol. 2, no. 3, pp. 202-211.

[5] X. Zhai and E. von Collani, "Strategies for Teaching a Novel Approach to Handling Uncertainty Scientifically via Internet", *International Journal of Emerging Learning*, 2009, vol. 4, no. 2, pp. 52-57.

[6] J. Bradley, M. Kemp, "The "Plus" Provided by Graphics Calculators in Teaching Undergraduate Statistics", 2nd *International Conference on the Teaching of Mathematics*, 2002, http://www.math.uoc.gr/ ictm2/

[7] M. Scaife, Y. Rogers, "External Cognition: How Do Graphical Representations Work?", *International Journal of Human-Computer Studies*, 1996, vol. 45, pp. 185-213. (doi:10.1006/ijhc.1996.0048)

[8] D. Tall, B. West, "Graphic Insight in Mathematics", in *The Influence of Computers and Informatics on Mathematics and its Teaching*, B. Cornu and A. Ralston, Eds. Paris, UNESCO, 1992, pp. 117-123.

[9] D. Tall, "Graphical Packages for Mathematics Teaching and Learning", in *Informatics and the Teaching of Mathematics*, D. C. Johnson and F. Lovis, Eds. North Holland, 1987, pp. 39-47.

[10] D. Tall, G. Sheath, "Visualizing Higher Level Mathematical Concepts Using Computer Graphics", *Proceedings of the Seventh International Conference for the Psychology of Mathematics Education*, Israel, 1983, pp. 357-362.

[11] C. Laborde, "Visual phenomena in the teaching/learning of geometry in a computer-based environment", *Perspectives on the Teaching of Geometry for the 21st Century – an ICMI Study*, M. Carmelo and V. Vinicio, Eds. 1998, pp. 113-121.

[12] K. Carolyn and M. Yerushalmy (Leaders), "The Working Group on Technological Environment", *The Future of the Teaching and Learning of Algebra The 12th ICMI Study*, K. Stacey, H. Chick and M. Kendal, Eds. 2004, pp. 97-152.

[13] K. Barry and T. Mike (Leaders), "The Working Group on CAS and Algebra", *The Future of the Teaching and Learning of Algebra The 12th ICMI Study*, K. Stacey, H. Chick and M. Kendal, Eds. 2004, pp. 153-166.

[14] http://www.gnuplot.info/

[15] http://www.java.com/

AUTHORS

X. Zhai has a Ph.D. in System Engineering in China and is presently a doctoral student of the Faculty of Mathematics and Computer Science, Würzburg University, Sanderring 2, D-97070 Würzburg, Germany. (e-mail: zhaixiaomin@ hotmail.com).

Dr. E. von Collani is a Professor at the Faculty of Mathematics and Computer Science, Würzburg University, Sanderring 2, D-97070 Würzburg, Germany. (e-mail: collani@mathematik.uni-wuerzburg.de).

Submitted 9 April 2009. Published as resubmitted by the authors on 9 August 2009.