# Creating user interfaces for econometric routines with JStatCom: An example for Gauss

Markus Krätzig

Humboldt-Universität zu Berlin, Institute for Statistics and Econometrics

mk@mk-home.de

## First version: 11 November 2004

#### Abstract

JStatCom is a new software framework that simplifies the creation of graphical user interface components for mathematical procedures. <sup>1</sup> It is written in Java and offers a coherent approach to creating applications for data based analysis. Programming with JStatCom is efficient, because existing algorithms written in various popular matrix languages can be reused with little or no changes required. The paper shows how this system can be applied to provide an existing Gauss program with a feature-rich graphical user interface with relatively little effort. One of the advantages of the presented approach is the strict separation of user interface and algorithm code. Although the framework provides standard solutions for common tasks, it can be extended and customized in various directions.

*Key words:* Java, Object-Oriented Programming, Gauss, Econometrics, Software Engineering

#### 1 Introduction

This text gives an introductory example on how to use the software framework JStatCom. The example demonstrates, how a graphical user interface for a statistical test could be implemented. This is a relatively simple but realistic scenario. The underlying algorithm is implemented in the Gauss programming language which is typical for econometrics. Readers who consider using the software framework for similar purposes are encouraged to follow the described steps and try it out.

The example uses the Eclipse Integrated Development Environment (IDE) which is an open-source project sponsored by IBM. This software is very popular in the

<sup>&</sup>lt;sup>1</sup> The URL is *www.jstatcom.com*.

Java community, also because it is freely available. But it should be stressed that this is just one tool among others and that the use of JStatCom is by no means related to any specific software development tool. Although not recommended, a simple text editor for Java coding would also be sufficient. Because this example should be reproducible, every step is documented including the setup of a project for the Eclipse IDE. Experienced Java developers can certainly skip those sections. Furthermore, it should be mentioned that the example is done under the Windows XP operating system.

## 2 A Step-by-Step Example

The example will implement a graphical user interface (GUI) for the ARCH-LM test (Engle, 1982) which provides the option to select a variable to test, as well as the number of lags to include. By pressing a button the test result should be computed and printed to a text area.

#### 2.1 System Requirements for this Example

- Windows 2000/XP
- working Gauss installation version 3.2 or higher

Although the Java part of JStatCom can be run on almost any platform, this demonstration requires an installed Gauss. The communication scheme between Gauss and Java has only been implemented for the Windows operating system. However, Gauss code could also be executed on other platforms with the help of the Gauss Runtime Engine (GRTE), which is a special Gauss distribution provided by Aptech. The GRTE allows for royalty free redistribution of compiled Gauss code that can be executed without an installed Gauss, thus enabling developers to ship standalone applications to their customers. JStatCom distinguishes between Gauss and the GRTE, although the two engines can execute the same Gauss code. But to develop with the GRTE one has to purchase a specific license first, therefore it would not be a very good candidate for this introductory example. At least, it is more likely to have a working Gauss installation available.

It should be mentioned, that the Gauss installation is only required to incorporate external code. But this example can also be run without this specific algorithm implementation. In this case the procedure for the test should be implemented in Java, which is also an option. Therefore readers are nevertheless encouraged to follow the remaining steps to get an idea about the workings of JStatCom.

Because Java code needs to be compiled and the created binary files (classes) must be executed with the Java Virtual Machine (JVM), it is necessary to download and install the current J2SE SDK from Sun, the URL is *www.java.sun.com.*<sup>2</sup> This software is also required to run the Eclipse IDE software, because it is written in the Java programming language as well. The installation follows a standard setup routine and should pose no problems.

## 2.3 Step 2: Download/Install the Eclipse IDE and some Plug-ins

As already mentioned, the Eclipse IDE is just a suggestion for use as a development tool. One of the great strengths of the Java platform is that there is excellent tool support. Therefore it would equally be possible to apply other tools with very similar steps.

First the IDE needs to be downloaded from *www.eclipse.org/downloads*. There are usually many different packages for various purposes available which might be a bit confusing. At the time of this writing the needed package is

eclipse-SDK-3.0.1-win32.zip. It is likely that the version number will already have changed when reading this text. The software just needs to be unpacked to some directory and will install itself when it is first started.

Because programming with JStatCom typically involves the layout of GUIs, a visual interface builder can speed up development significantly. Such a tool is not part of the standard Eclipse package, but can be installed as a plug-in. The following additional packages need to be downloaded from

*www.eclipse.org/tools.* The names of the relevant archives are given in brackets, although the version numbers are likely to have changed already:

- EMF (emf-sdo-runtime-2.0.1.zip)
- GEF (GEF-SDK-3.0.1.zip)
- VE (VE-runtime-1.0.1.1.zip)

Installing those plug-ins merely requires to unpack the archives to the Eclipse installation directory and to restart the IDE. All mentioned tools can be downloaded free of charge.

 $<sup>^{2}</sup>$  The most recent release is J2SE 5.0. With this version the term SDK (Software Development Kit) changed to JDK (Java Development Kit).

The last required download is the JStatCom software framework itself, which can be found under *www.jstatcom.com*. The archive jstatcom\_win-1.0.zip just needs to be extracted to some folder. By default it creates a directory jstatcom. It is recommended to rename this directory to something more descriptive for the respective project. For the current example the name testproject is used.

#### 2.5 Step 4: Create an Eclipse Project

New Project elect a wizard				<u></u>
Create a Java project				
Vizards:				
Java Project				
∃ / / / / / / / / / / / / / / / / / / /	work			
🗄 🔁 Plug-in Development				
Hings Simple				
				(2)
	Conserve 1	1	1	11
	< Back	Next >	Finish	Cancel

Fig. 1. Selecting a new Java project in the Eclipse IDE

In the following it is assumed that JStatCom is located in the directory

c:\testproject. Now the Eclipse software should be started, the option to change the workspace directory can be ignored. At first startup a help screen is presented, it can be closed. The first step should be to create a new Java project via *File* -*New*. The resulting dialog is presented in Figure 1. When *Next* is pressed Figure 2 is shown where the project name should be given, as well as the directory with the resources. The created project can then be seen in the package browser, see Figure 3. All jar archives in the project directory have automatically been included in the build class path, for example jstatcom.jar, jama.jar, junit.jar, etc.. By default, all classes that will be compiled go in a subdirectory bin. But no classes have been created so far.

eate a Java project				
reate a Java project in the v	vorkspace or in an exte	rnal location.		Ĵ
roject name: testproject				
Location				
C Create project in works	pace			
Create project at exter	nal location			
Directory: C:\testproject			B	rowse
Project layout				
🖲 Use project folder as ro	ot for sources and clas	s files		
C Create separate source	e and output folders			
			Configur	e Defaults
	on already exists. If a p	roject is created	d in this location onfigure the cla	n, the wizard asspath
The specified external locatic will automatically try to deter appropriately.				

Fig. 2. Setting project name and directory



Fig. 3. Display of the new project testproject in the package explorer 2.6 Step 5: Create a new Package and a new Class

Programming with Java usually starts with creating a class. The class to create for this example should hold the GUI for the test and should itself be a so called internal



Fig. 4. Creating a new class

- Gaussanio_u.jar	🚝 New Java I	Package	2
j=ou jace.jar I=ō jama.jar I=ō jhall.jar I=ō junit.jar	<b>Java Packag</b> Create a Java	e package.	Ť
i jstatcom.jar	Creates folders	corresponding to packages.	
JRE System Library []2781.4.2_03] 	Source Folder:	testproject/src	Browse
doc jars	Name:	com.myorg	
igauss			
l 🥭 jgrte			
iniau Inice jox			
🗁 jstub			
🗠 🥭 testdata			
- Contract of the second secon			
app.bat			
app.properties			
	<u></u> 88		
EN DI LA DA DA DA DE DE			

Fig. 5. Creating a new package

frame. This is a special component that is displayed as a window within a desktop application.

Package Explorer × Hierarchy	- 0)		OUST THE
← → Q   E	New Java Class		×
🕀 🔛 JHelpDev	A series surre class		
🕀 😂 jmulti	Java Class		~
🕀 😂 jstatcom	Create a new Java	class.	C
🖻 🔛 testproject			
🕀 🧕 concurrent.jar			
🕀 🧕 GaussAPI5_0.jar			
🕀 💽 jade.jar	Source Folder:	testproject/src	Browse
🕀 🚺 jama.jar	Dackages		Browce
🖽 😡 jhall.jar	rauwaye,	Teomanyorg	Browsen
🖽 👩 junicijar	Enclosing type:		Browse
🖽 📷 jxl.jar			
III III JDE Suctorn Library [i2ro1 4 2 02]			
General Second Contrary []2re1.4.2_03]	Name:	TestFrame	
	Modifiers:	public C default C private C protected	
		abstract     final     static	
iars			
H 🔁 igauss	Superclass:	com.jstatcom.model.ModelFrame	Browse
🕀 🦕 jarte	Interfaces:		http://www.alia
🗄 👝 jmlab			Hadan
🗄 🔁 jox			The second secon
🗄 🗁 jstub			Remove
🗄 🗁 testdata	tokak sankadarak	and the second second second	
🗄 🗁 unittests	which method stud:	s would you like to create?	
MANIFEST.MF		I public static void main(String[] args)	
README.txt		Constructors from superclass	
app.bat		✓ Inherited abstract methods	
app.properties			
一派] build.xml			
disp_win_1386.dll			
meip_context.xml			
jgauss_gcg.bat		Finish	Cancel
jgauss_lib.bat			

Fig. 6. Specifying the new class

🛱 Package Explorer 🖾 🛛 Hierarchy 💭	D TestFrame.java ×
+ -> 6   E 😫	👻 📕 package com.myorg;
DrelpDev     JrelpDev     jondi     jondi     concurrent.jar     GaussAP15_0.jar     jona.jar     jona.jar     jonuli.jar     jolul.jar     jolul.jar	<pre>import com.jstatcom.model.ModelFrame;  */** * * Calculation &lt;= href="mailto:mk@mk-home.de"&gt;Markus Kraetzig */ * public class TestFrame extends ModelFrame ( )</pre>

Fig. 7. Generated Java code for a new class

Before a new class can be created, the IDE software requires to define a directory with the source code to compile. It will look for Java source files only there and compiles them when changes are made. Figure 4 shows the menu that appears on a right mouse click over the project name testproject. By selecting *Source Folder* a small dialog will appear asking for the directory name which can freely be chosen, but is typically src.

Afterwards the same menu should be used over the just created src folder, but this time with the option *Package* to create a new Java package. The dialog for this is shown in Figure 5. It is not strictly required to define packages, but it is good programming practice. The standard for this is the reverse domain name of

the organization or firm that develops the Java code. Here com.myorg was used as a placeholder. As a starting point one could also use a descriptive name, like testproject. It can be changed at a later stage easily.

Now everything is prepared to create the first Java class. Again, the right mouse menu should be used, this time over the newly created package

src/com.myorg with the *Class* option. Figure 6 shows the dialog to specify the class to create. The name of the class can freely be chosen, but class naming conventions for Java suggest that it should consist of nouns, each of them beginning with a capital letter, for example TestFrame. As a second step it is important to choose the superclass that the new class inherits from. By default this is just Object, the top-level superclass of all classes in Java. However, some more functionality is needed because the class should be a frame holding the GUI for the test. Therefore one should select the class ModelFrame with the help of the *Browse* button in the *superclass* field. It is then necessary to remove the default Object entry first and type in the first letters of ModelFrame. The dialog will immediately display all possible options. Figure 6 shows the result. All other options can be left unchanged.

After finishing the dialog, a Java editor window with the code for the newly created class is shown, see Figure 7. Also, the new file is added to the package browser in the source directory src under the Java package name com.myorg. The class TestFrame is a subclass of ModelFrame, which is a component from the JStat-Com system. The generated code is very short, because the new class does not yet define own methods or fields. It inherits all functionality from its superclass. One should consider documenting the header of TestFrame with some remarks about the function of this component. In the example, only the author is mentioned.

#### 2.7 Step 6: Layout the GUI with the Visual Editor

> doc	New	•	
jars jasuss	Open	F3	
jgauss jarte	Open With		<ul> <li>Java Editor</li> </ul>
🤊 jmlab	Open Type Hierarchy	F4	💫 Visual Editor
jox istub	Сору	Ctrl+C	Text Editor
testdata	Paste	Ctrl+V	
unittests	🕻 Delete	Delete	Displace Editor
DEADME HYP	Source	Alt+Shift+S	Default Editor
ann bat	Refactor	Alt+Shift+T 🕨 🕇	Derault Editor

Fig. 8. Open class in visual builder tool



Fig. 9. Initial display, frame needs to be resized

After having set up the skeleton of the new module, the GUI should be laid out with the help of the visual editor (VE) that has been installed via the mentioned plug-ins.

To use the VE tool in Eclipse, it is first necessary to close the Java editor window that is currently showing TestFrame.java. Afterwards the same file should be

opened again in the VE by right clicking over the file TestFrame.java in the package explorer and selecting *Visual Editor*, see Figure 8.

After invoking this operation, it might take a while until the VE is set up, because this is a computationally intense operation. The initial display is shown in Figure 9. There is an area at the top where the actual state of the GUI is shown as it would look like during runtime. The frame is displayed with its minimal size in the beginning. The Java editor underneath shows the code that is generated by the VE, which is nothing so far. After all, the VE is just a code generating tool and it is also possible to do manual adjustments by editing the Java code directly. On the right edge there is the so called *Palette* which provides access to many standard Java components that can be picked and placed in the VE. It can also be used to access all components provided by the JStatCom framework, although those are not displayed with an icon but must be chosen via the button *Choose Bean* from the palette.

Having introduced the basic features of the VE, editing can be started by resizing the frame with the mouse and choosing a title. The title can be set by clicking in the header of the frame, or by directly editing the properties of that component. For that purpose, there exists the property editor of the VE, see Figure 10 at the bottom. It should be noted, that most properties do not need to be changed from their default value, but the title property has been set to TestFrame, lacking more innovative ideas here. By changing the size of the frame with the mouse, the size property is automatically changed. Properties are used to customize components and help to speed up programming, especially for GUI layout.

Now that the TestFrame has a title and a proper size, it is required to add a panel to hold all other components. Therefore one should select the class JPanel from the palette and place it in the frame. Because panels are container elements, they can be found under the *Swing Containers* tab, see Figure 11. Adding a JPanel is not a very exciting operation, because visually nothing changes. It is just an empty container. However, by looking at the Java editor one can see that some code has been generated and the property editor is now showing the properties of the added JPanel.

Now a small technicality is required to avoid confusions when adding new components to the panel. By default every panel has a layout manager which computes how components are being placed inside the container. It also adjusts their size and placement when the container is resized. The default manager for the JPanel is a FlowLayout. However, for this example layout management should not be discussed and it is easier to switch it off for the initial design. Therefore the property editor should be used to set the layout property of this panel to null as shown in Figure 12.

Having prepared the panel to hold the components, one should now select the class

TSSel from the palette via the *Choose Bean* button. Figure 13 shows the selection dialog. After clicking OK the component can be placed on the panel inside the frame. If the enclosing frame is still not big enough, it should be resized again. After finishing this operation the new component is part of the TestFrame, see Figure 14.

The chosen component is part of the JStatCom framework and is used to select variables for time series models. It provides also a range of useful functions via a right click popup menu over the selected variables. Furthermore, the sample range can be adjusted. This way, complex functionality is integrated in the testing frame which demonstrates the advantages of component based development.

However, there are some properties that need to be set to the TSSel component with the help of the component editor. Because the class is used to select variables, it is necessary to define under which name the data objects are stored in the so called symbol table. JStatCom provides a mechanism to share data between components, a feature that is, for example, also available for Matlab GUI building in a somewhat similar way with the Matlab guidata function. This is only mentioned to show that similar problems arise also in other systems for GUI programming. For JStatCom, data objects must be identified via names in the symbol table. Therefore the property editor should be used to set these names, see Figure 15. When the TSSel component is selected in the VE one can start editing the properties allDataName, allStringsName, and dateRangeName. The chosen names for this example are Y, Y\_NAMES, and DATE\_RANGE respectively.

After finishing the setup of the selection component, one still needs to add the remaining components. JStatCom provides the special text field NumSelector to select numbers. It should be selected and placed to the panel in the same manner as the TSSel component, see Figure 16.

One of the features of the number selection component is that it is easily possible to set a validating range with the help of the property editor. From Figure 17 it can be seen how the property rangeExpr is set to the string [0, 20]. Whenever the user specifies a lag length outside this range, an error message would appear. Furthermore, the precision of the display is set to 0 by default, which means that only integer values are shown with no decimal digits.

By now it should be clear how components can be added to the GUI. The remaining parts are a JLabel with the string Lags, a JButton with Execute on it, and a ResultField to display results. The latter component is provided by JStatCom, whereas the first two are standard Swing classes and can be found directly on the palette under Swing Components. All components should resized and placed according to the preferences of the developer. The text of the JLabel and the Execute button can be set by clicking on the component with the mouse or by using the property editor and changing the text property. One should not forget to save the file TestFrame.java after adding components. This can easily be done by pressing the Ctrl-S key combination. Saving a Java source file in the IDE automatically invokes the compilation.

TestFrame		5 <del>77</del>
TestFrame	<u>*</u>	Palette
		Palette Selection Selecti
<pre>package com.myorg; import com.jstatcom.mode \$\approx / ** * * * * * * * * * * * * * * * * *</pre>	el.ModelFrame; to:mk@mk-home.de">Markus Kraetzig	Jist Just JComboBox JTree JTable on JScrollPane JTable on JScrollPane JTable JTable TableColumn JProgressBar JOptionPane JTextPane JEditorPane
		_ I 😕 Swing Menus
*7	vtends ModelFrame (	
*/ ♥mmhlic class TestFrame e	Þ	AWT Controls
Toblems Javadoc Declaration	ies 🕅 Java Beans	AWT Controls
roblems Javadoc Declaration Property	ies 🕄 Java Beans	AWT Controls
roblems Javadoc Declaration Property background	ies 🕃 Java Beans Value 204,204,204	AWT Controls
Troblems Javadoc Declaration Properti Property background bounds	ies ⊠ Java Beans Value 204,204,204 0,0,551,295	AWT Controls
Property background bounds closable	Value           204,204,204         0,0,551,295           true         true	AWT Controls
roblems Javadoc Declaration Property background closable closed	Value           204,204,204         0,0,551,295           true         false	AWT Controls
Property background closable closable componentOrientation	Value           204,204,204         0,0,551,295           true         false           UNKNOWN         UNKNOWN	AWT Controls
Troblems Javadoc Declaration Property background bounds closable closed componentOrientation default/loseOperation	Value           204,204,204         0,0,551,295           true         false           UNKNOWN         DISPOSE ON CLOSE	AWT Controls
roblems Javadoc Declaration Property background closable closable closed componentOrientation defaultCloseOperation enabled	Value  Value  204,204,204  0,0,551,295  true  false  UNKNOWN DISPOSE_ON_CLOSE  true	AWT Controls
Troblems Javadoc Declaration Property background bounds closable closed componentOrientation defaultCloseOperation enabled foor	Value  Value  204,204,204 0,0,551,295 true false UNKNOWN DISPOSE_ON_CLOSE true Dialog_plain_12	AWT Controls
Property background closable closable closable closable closable closable closable closable closable closable closable closable closable closable closable closable closable closable closad componentOrientation defaultCloseOperation enabled font foreground	Value Value 204,204,204 0,0,551,295 true false UNKNOWN DISPOSE_ON_CLOSE true Dialog, plain, 12 Color:black	AWT Controls
Toblems Javadoc Declaration Property background bounds closable closable closed componentOrientation defaultCloseOperation enabled font foreground foreground framefron	Value           204,204,204         0,0,551,295           true         false           UNKNOWN         DISPOSE_ON_CLOSE           true         Dialog, plain, 12           Color:black         Color:black	AWT Controls
Property background bounds closedle closed closedle closedle componentOrientation defaultCloseOperation enabled font foreground frameIcon iconfibile	Value       204,204,204       0,0,551,295       true       false       UNKNOWN       DISPOSE_ON_CLOSE       true       bialog, plain, 12       Color:black	AWT Controls
Toblems Javadoc Declaration Propert Property background bounds closable closed componentOrientation defaultCloseOperation enabled font foreground frameIcon iconifiable becktion	Value       204,204,204       0,0,551,295       true       false       UNKNOWN       DISPOSE_ON_CLOSE       true       Dialog, plain, 12       Color:black       true       0.0	AWT Controls
Toblems Javadoc Declaration Property background bounds closable closed componentOrientation defaultCloseOperation enabled font foreground frameIcon iconifiable location mandriable	Value           204,204,204         0,0,551,295           true         false           UNKNOWN         DISPOSE_ON_CLOSE           true         Dialog, plain, 12           Color:black         true           true         0,0	AWT Controls
Toblems Javadoc Declaration Property     Declaration Property     Declaration Property     Declaration Property     Declaration Property     Declaration Property     Declaration     Closable     closable     closable     closed     componentOrientation     defaultCloseOperation     enabled     font     foreground     frameIcon     iconifiable     location     maximizable     restmente	Value           204,204,204         0,0,551,295           true         false           UNKNOWN         DISPOSE_ON_CLOSE           true         true           bialog, plain, 12         Color:black           true         folse           true         folse	AWT Controls

Fig. 10.  ${\tt TestFrame}$  after resize with title, properties at the bottom

🔒 *TestFrame.java 🗙			- 8
			Palette
TestFrame		r <sup>k</sup> ⊠	Palette   Palette  Palette  Selection  Marquee  Choose Bean  Swing Components  Swing Components  Swing Containers  Swing Containers  Supare  JFrame JFrame JScrollPane JSplitPane JTabbedPane JTabbedPane JInternalFrame JDialog JWindow JToolBar JJoolBar JApplet
<pre>private JPanel ge if (jPanel == jPanel = n ) return jPanel } } // @jve:decl-indu</pre>	tJPanel() ( null) ( new JPanel(); ; ex=0:visual-constraint="10,10"	-	
<pre>private JPanel ge if (jPanel == jPanel = ) ) return jPanel ) ) // @jve:decl-ind</pre>	tJPanel() ( null) ( new JPanel(); ; ex=0:visual-constraint="10,10"	-	Swing Menus
<pre>private JPanel ge if (jPanel == jPanel = ) ) return jPanel ) ) // @jve:decl-indu ]</pre>	tJPane1() { null) { new JPanel(); ; ex=0:visual-constraint="10,10"	• • •	Swing Menus AWT Controls
<pre>private JPanel ge if (jPanel == jPanel = n ) return jPanel } // @jve:decl-inde * Yroblems Javadoc Declaration Prop</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties 22 Java Beans	 ₹	Swing Menus AWT Controls
<pre>private JPanel ge if (jPanel == jPanel = n ) return jPanel } } // @jve:decl-inde * Yroblems Javadoc Declaration Property</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties 22 Java Beans Value	• •	Swing Menus AWT Controls
<pre>private JPanel ge if (jPanel == jPanel = 1 } return jPanel } // @jve:decl-indu ?roblems Javadoc Declaration Property background</pre>	tJPane1() { null) { new JPane1(); ; ex=0:visual-constraint="10,10" perties 2 Java Beans Value 204.204.204	• • •	Swing Menus
<pre>private JPanel ge if (jPanel == jPanel = 1 ) return jPanel } ) // @jve:decl-ind roblems Javadoc Declaration Property background border</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties 23 Java Beans Value 204,204,204 EmotV(0.0.0.0)		Swing Menus
<pre>private JPanel ge if (jPanel == jPanel = 1 ) return jPanel } } // @jve:decl-indu ? Problems Javadoc Declaration Property background border componentOrientation</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties  Value 204,204,204 Empty(0,0,0,0) UNKNOWN		Swing Menus AWT Controls
Image: system of the system	tJPanel() { null) { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties 2 Java Beans Value 204,204,204 Empty(0,0,0,0) UNKNOWN brue	• • •	Swing Menus AWT Controls
<pre>private JPanel ge if (jPanel == jPanel = 1 ) return jPanel } ) // @jve:decl-indu roblems Javadoc Declaration Prop Property background border componentOrientation enabled &gt;field name</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10"  perties 23 Java Beans  Value 204,204,204 Empty(0,0,00 UNKNOWN true iDenel	▲ 	Swing Menus AWT Controls
<pre>private JPanel ge if (jPanel == jPanel = 1 ) return jPanel } } // @jve:decl-inde ? Problems Javadoc Declaration Property background border componentOrientation enabled &gt;Field name feat</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties  Value 204,204,204 Empty(0,0,0) UNKNOWN true jPanel Dides align 12	- 	Swing Menus AWT Controls
Image: system of the system	tJPane1() { null) { new JPane1(); ; ex=0:visual-constraint="10,10" perties 2 Java Beans Value 204,204,204 Empty(0,0,0,0) UNKNOWN true jPanel Dialog, plain, 12 indog, plain, 12	• • •	Swing Menus
<pre>private JPanel ge if (jPanel == jPanel = 1 ) return jPanel } } // @jve:decl-indu ] Property background border componentOrientation enabled &gt;field name font foreground</pre>	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10"  value 204,204,204 Empty(0,0,0) UNKNOWN true jPanel Dialog,plain,12 Color:black		Swing Menus AWT Controls
▼       private JPanel ge         if (jPanel ==         jPanel = n         jPanel = n         )         return jPanel         )         return jPanel         )         // @jve:decl-inde         )         Problems         Javadoc       Declaration         Property         background         border         componentOrientation         enabled         >field name         fort         foreground         layout	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties 🖾 Java Beans Value 204,204,204 Empty(0,0,0,0) UNKNOWN true jPanel Dialog, plain, 12 Color:black FlowLayout		Swing Menus AWT Controls
▼       private JPanel ge         if (jPanel ==         jPanel = n         jPanel = n         )         return jPanel         )         return jPanel         )         // @jve:decl-inde         )         Problems         Javadoc       Declaration         Property         background         border         componentOrientation         enabled         >field name         font         foreground         * layout         name	tJPane1() { null) { new JPane1(); ; ex=0:visual-constraint="10,10"  perties 2 Java Beans Value 204,204,204 Empty(0,0,0,0) UNKNOWN true jPanel Dialog, plain, 12 Color:black FlowLayout	• • •	Swing Menus
▼       private JPanel ge         if (jPanel =         jPanel =         jPanel =         )         return jPanel         )         // @jve:decl-induced         )         // @jve:decl-induced         voblems       Javadoc         Declaration       ■ Property         border       componentOrientation         enabled       > field name         foreground       1         ± Jayout       name         preferredSize       1	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10"  value 204,204,204 Empty(0,0,0) UNKNOWN true jPanel Dialog,plain,12 Color:black FlowLayout 10,10		Swing Menus AWT Controls
▼       private JPanel ge         if (jPanel ==         jPanel = n         jPanel = n         )       return jPanel         )       return jPanel         )       // @jve:decl-inde         )       jPoblems         Javadoc       Declaration         Problems       Javadoc         Declaration       ● Property         background       border         componentOrientation       enabled         >field name       fore         foreground       Elayout         name       preferredSize         toolTipText       Elayout	tJPanel() { null) { new JPanel(); ; ex=0:visual-constraint="10,10" perties S Java Beans Value 204,204,204 Empty(0,0,0,0) UNKNOWN true jPanel Dialog, plain, 12 Color:black FlowLayout 10,10	• •	Swing Menus AWT Controls

Fig. 11. TestFrame after a JPanel has been added

Problems Javadoc Declaration 🔲 Properties 🕱	Java Beans	E 🐉 🗔 🗣 🔻
Property	Value	
background	204,204,204	
border	Empty(0,0,0,0)	
componentOrientation	UNKNOWN	
enabled	true	
>field name	jPanel	
font	Dialog, plain, 12	
foreground	Color:black	
🖃 layout	null	
alignment	null	
horizontal gap	BorderLayout	
vertical gap	BoxLayout(X_AXIS)	
name	BoxLayout(1_AXIS)	-
preferredSize	10,10	131.VI
ke el Tie Teuk		

Fig. 12. Setting layout manager to null, easier to handle for initial design

*TestFra	듣 Choose a Bean		1
Tes	Select class Choose a JavaBean (? = any character, * = a	any string)	Palette Selection Marquee
	Matching types:		Choose Bean
	R TCC-1		Swing Components
			JFrame JPanel JScrollPane JSplitPane JJabbedPane JJabkdPane JJsktopPane JInternalFrame
			1Window
	Qualifier:	jar	JVindow
× 1	Qualifier:  Com.jstatcom.ts - /testproject/jstatcom.  C All Types C Swing C AWT	jar	JoolBar Bar Bar JApplet
× 1	Qualifier:	jar	JoolBar JoolBar JoolBar JoolBar
× 1	Qualifier:  Qualifier:  Com.jstatcom.ts - /testproject/jstatcom.ts  All Types C Swing C AWT  Properties Bean variable name TSSel	jar	J J Window

Fig. 13. Selecting the TSSel component

🚯 *TestFrame.java 🗙		
Teatrons .	_ <b>×</b> → 1	Palette 🔤 🕨
		Selection
	Ctrl/Shift for Multiple Selection-	L_4 Marquee
		So Choose Bean
		Swing Components
		Swing Containers 🖈
		JFrame
		JPanel
		JScrollPane
		JSplitPane
		JTabbedPane
		JDesktopPane
		JInternalFrame
	[1960.01.1982.04	JDialog
	Max Confirm Selection	JioolBar
	Selection Order sets Variable Order	
	**	
jPanel = ne	w JPanel();	-
jPanel.setL	ayout (null);	Co Suina Manur
JPanel.add(	gecissei(), null);	
	<u>N</u>	AWI CONCROIS

Fig. 14. TSSel component has been placed on the panel

Property	Value
>allDataName	Y
>allStringsName	Y NAMES
background	204,204,204
(+) border	Title("Ctrl/Shift for Multiple Selection", Right, Top. (dialog, plain, 10), (Color:black))
>bounds	309,11,216,400
checkNaN	true
componentOrientation	UNKNOWN
>dateRangeName	DATE_RANGE
datesEditable	true
deterministicDataName	
deterministicEnabled	true
deterministicStringsName	
enabled	true
endogenousDataName	
endogenousStringsName	

Fig. 15. Editing properties of TSSel component

TestFrame.java 🗙 🔪	🚰 Choose a Bean	
TestFrame	Select class Choose a JavaBean (? = any character, * = a NumSel Matching types: NumSelector	ny string)
	Qualifier: com.jstatcom.component - /testproject/js	tatcom.ja
	All Types C Swing C AWT Properties Bean variable name numSelector	
j] j1 Million Javadoc Declara	com, jstatcom, component. NumSelector	Cancel

Fig. 16. Selecting the NumSelector component for number input

Color:black LEADING false 58,52 0				
Color:black LEADING false 58,52 0				
LEADING false 58,52 0				
false 58,52 0				
58,52				
0				
0				
(numberFormatTypes\$1)				
(numberRange)				
0				
11,20				
[0,20]				
55,20				
(numberRange) 0 11,20 [0,20] 55,20				

Fig. 17. Setting a validating range [0, 20] to the NumSelector component



Fig. 18. Placing a JButton and creating an action



Fig. 19. Default action handler for execute button

After the basic layout of the GUI has finished, it is necessary to program an action that is invoked when the *Execute* button is pressed. Without this, nothing would happen at all. The VE also supports this task via a menu that appears on a right click over the button, see Figure 18. The item *Events - actionPerformed* should be chosen to install a default action to that component.

Figure 19 shows the Java code that has been generated by this operation. There is an actionPerformed method which just prints out some default string. This method

must later be programmed 'by hand' to gather the input from the GUI controls, to call the external Gauss procedure, and to print the results.

## 2.9 Step 8: Add the Module to the Main Application Frame



Fig. 20. Editing modules.xml to insert TestFrame to list of modules



Fig. 21. Adjusting the classpath in the app.bat script

Before it is described how the algorithm for the ARCH-LM test is invoked one would probably like to check what has been accomplished so far. Therefore the module is integrated in the JStatCom application framework and can be run. For this, no further Java programming is required, only two configuration files need to be adjusted.

Figure 20 shows the file modules.xml in the text editor of the IDE. It should be opened via the right mouse menu. The format of that file is XML and it already contains some example entries which have been commented out. To integrate the newly created module, the line

<Module class="com.myorg.TestFrame"/>

<b>]</b> ] ]	StatCom						
File	Modules Windows Control	Help					
100	🐇 TestFrame						
			TestFrame				
					Ctr	I/Shift for Multiple Selection	-
		a (1,24)			P.		-
	🔄 Import Dataset		0	Lags			
	Filename	· · · ·					
	Description			Execute			SET
	Description	L T					
	10 ANNO 11						
	Loaded Data	-					
1.3					1960 Q1, 196	2 Q4	
					Max	Confirm Selection	
1							
					Selectio	on Order sets Variable Order	
80							
3					1/107.		
	<u> </u>						
							STOP

Fig. 22. Running application with new TestFrame module, execute method still needs to be coded

should be added and the file should be saved. Note that the slash at the end is needed. Afterwards the file app.bat has to be edited to add the location of the classes to the classpath, see Figure 21. The folder with the compiled classes is bin by default, therefore the line

SET CP = bin;%CP%

must be added. Finally, one can now start the application by clicking on the file app.bat from outside the IDE software. It will invoke the application and the module can be accessed via the menu item *Modules*. Figure 22 shows the running program. Although nothing happens yet when *Execute* is clicked, except that some string is printed to the standard output, there are already a lot of features available provided by the framework.

For example, it is possible to import datasets, to use the time series calculator for combining variables with arithmetic operations, and to edit and transform series that have been read in. It is also possible to create dummy variables conveniently. In the module TestFrame one can select variables, and the text field for the number of lags validates input according to the interval set. All this general functionality is provided by JStatCom and it is now up to the developer to use this infrastructure for implementing calls to specific algorithms.

After having programmed the GUI in Java, one has to prepare the Gauss code with the algorithm. JStatCom provides a communications interface to an installed Gauss, but the code has to be compiled as a gcg file first and it has to be put in a special location. The Gauss procedure to compute the ARCH-LM test is given in the code example. If one is only interested in the general workings of this example, one could also use some shorter procedure or return just a number.

```
/**
 * ARCH-LM test (Engle 82).
 * resid - Tx1 vector with residuals
 * q - number of lags to include in test regression
 * result - Chi2 stat prob chi2 F stat prob F
 */
proc(1) = archlm mytest(resid, g);
  local y,ylags,Tnew,b,sigma,rsq,f,XX,r,prob_chi,prob_F;
          = (resid - meanc(resid))^2;
   y
   ylags = shiftr((ones(1,q) .*. y)', seqa(1,1,q), -exp(20))';
   ylags = delif(ylags,ylags[.,cols(ylags)] .== -exp(20));
    Tnew = rows(ylags);
          = y[rows(y)-Tnew+1:rows(y)];
    У
   ylags = ones(rows(ylags),1)~ylags;
   b = inv(ylags'ylags)*ylags'*y;
    sigma = (y-ylags*b) ' (y-ylags*b) / rows(y);
    rsq
          = 1-sigma/((y - meanc(y))'(y - meanc(y))/rows(y));
   XX
          = ylags'ylags;
           = zeros(q, 1) ~ eye(q);
   R
          = (R*b)'inv(R*inv(XX)*R')*(R*b) / (q*sigma);
   F
   prob_F = cdffc(F,q,rows(ylags)-rows(b));
   prob chi= cdfchic(rows(ylags)*rsq,q);
retp(rows(ylags)*rsq~prob_chi~F~prob_F);
endp;
```

It is assumed that the code is stored in the file mytest.src. This file should be put in the subdirectory jgauss/src of the project directory. There are also some other source files which are needed by the communications library. One should be aware that Gauss procedure names should be unique across all files, therefore the name archlm\_mytest was chosen. The convention is, that the filename is used as a suffix to the descriptive name of the procedure.

As a last step, one must compile the source code together with the other files in the directory to a file jgauss.gcg. This could be done manually with Gauss, but JStatCom provides the batch script jgauss\_gcg.bat to make it easier to compile

also a large number of source files. For this the file jgauss/compile.xml has to be edited:

Only the last line has been added. By clicking on jgauss\_gcg.bat all source files are compiled and the file jgauss/jgauss.gcg is being created. When the batch script is started first it is likely that some settings for the Gauss software have to be set. Those settings could also be set manually in the file

jgauss/engine\_config.xml. JStatCom needs to know the correct version number and the location of the Gauss executable.

#### 2.11 Step 10: Implement the Execute Routine



Fig. 23. Open TestFrame in Java editor, more convenient for manual coding than visual editor

Now that the Gauss procedure archlm\_mytest is recognized by the Gauss engine,



Fig. 24. Hidden method body that should be expanded by clicking on arrow to the left



Fig. 25. Red underline and symbol to the left indicating some compilation problem



Fig. 26. A left mouse click on the error symbol gives a menu with possible error fixes, the first option should be chosen here

it is possible to implement the Java call to that procedure. It should be mentioned that JStatCom has its own data model to represent numbers, matrices, strings, string arrays, dates, and date ranges. All data classes start with the prefix JSC, which is short for JStatCom. For example, JSCNArray represents an  $m \times n$  number array, whereas JSCInt stores an integer scalar. The basic steps for the execute call are:

```
(1) Get Gauss engine instance
Engine engine = EngineTypes.GAUSS.getEngine();

(2) Retrieve the input data objects (// starts a Java comment)
// retrieves shared symbol by name
JSCNArray y = global().getSymbol("Y").getJSCNArray();
```



Fig. 27. Implementation of the execute call

// creates new integer object with the selected lags
JSCInt lags = new JSCInt("LAGS", getNumSelector().getIntNumber());

- (3) Create an empty data object for the result of the computation
   JSCNArray result = new JSCNArray("TESTRESULT");
- (4) Call the Gauss procedure

// call the procedure with input and output parameters
engine.call("archlm\_mytest", new JSCData[]{y, lags},

new JSCData[]{result});

(5) Display the result and set it to the symbol table
 getResultField().append(result.display());
 global().set(result);

Some manual Java programming is required here. This is done more easily with the Java editor instead of the visual editor in the IDE. Therefore the VE should be closed and the Java editor should be opened via the right click menu over the TestFrame.java file, see Figure 23. It should be noted that by default this method is hidden in the editor and should be expanded first, see Figure 24. When editing is started, one could use the automatic expansion mechanism by pressing the Ctrl-Space keys together. If any typos occur, the editor indicates this by underlining the respective element and showing an error symbol at the left edge, see Figure 25. Often problems occur because a class name is only recognized after it has been imported. However, by clicking on the error symbol one can easily fix those problems automatically, see Figure 26. The full method body is seen in Figure 27.



Fig. 28. Running module with output from computation

Symbol Control							
TESTFRAME	Name	TES NAR	TRESULT RAY				———Description –
	Row	rs 1 4		Save		Save Options	
	1	15.4368	2	3 9.3381	4 0.0002		

Fig. 29. Symbol Control after the computation has finished

Now that everything has been programmed, the module should be checked again via clicking on the app.bat script. After a series has been imported and selected, a click on the *Execute* button will invoke the computation and the result will be printed to the text area. The resulting numbers include the Chi-squared and F statistics together with the corresponding p-values, see Figure 28. Furthermore, the re-

sult has been set to the symbol table of the module. This can be checked with the Symbol Control which is accessible via the *Control* menu, see Figure 29.

Another small customization of the new application could be done by editing the file app.properties in the project directory. It is possible to change the title, the splash screen, the version number, as well as the about information of the software.

#### 3 Final Remarks

This example should have motivated the use of JStatCom, although it has only scratched the surface of what can be done with the framework. The module that has been programmed could still be refined in several ways. There should be an input check before the procedure call is actually carried out. Output formatting could be improved as well. Furthermore, one should use a layout manager to make the GUI look more professional, especially when it is being resized. Another important point is that the module consists of only one class, which is appropriate for this simple example, but which should be changed if more features are to be added. In that case one should consider creating a separate class which only holds the panel. It could then be added to the frame. If more features are implemented, the frame could get a tabbed pane or a menu bar, and it could hold many panels instead of just one.

An important point is that the execute routine in its current form is defined inside the GUI class. In general this is not recommended and JStatCom provides the so called PCall system to encapsulate the calling logic in separate classes. This has the advantage that the procedure calls could be maintained separately and that they could also be used independently of the GUI classes. However, having this introductory example running is a good starting point for many possible enhancements.

#### References

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity, with estimates of the variance of United Kingdoms inflations, *Econometrica* **50**: 987–1007.