Programming of a Web Interface for the Database of the First G-APD Cherenkov Telescope for a Validation of the Data Quality

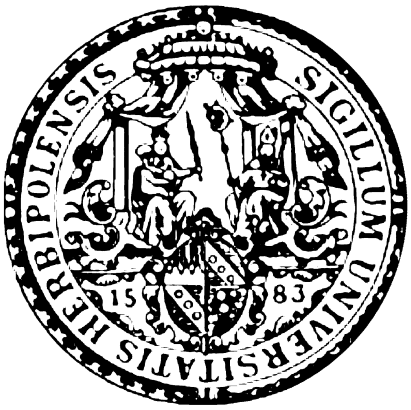
FACT (Credit: Miguel Claro)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Bachelor Thesis
Christina Hempfling

Programming of a Web Interface for the Database of the First G-APD Cherenkov Telescope for a Validation of the Data Quality

Bachelor Thesis



Julius-Maximilians-Universität Würzburg
Faculty of Mathematics and Computer Science
Course of Studies: Aerospace Computer Science

written by
Christina Hempfling

Würzburg, 2013

Die vorliegende Arbeit wurde im Zeitraum vom 14.10.2013 bis zum 23.12.2013
am Lehrstuhl für Astronomie der Julius-Maximilians-Universität Würzburg angefertigt.
Prüfer: Prof. Dr. Matthias Kadler, Prof. Dr.-Ing. Hakan Kayal
Betreuerin: Dr. Daniela Dorner

Abstract

The First G-APD Cherenkov Telescope (FACT) is located on the Canary Island La Palma, Spain. It is the first Cherenkov telescope to use special photosensors, so called Geiger-mode avalanche photodiodes (G-APDs), to monitor active galactic nuclei (AGN) in the TeV-energy band. Since it is an earthbound telescope, the data taken can be affected by varying night sky background light such as the moon, by weather conditions (e.g. clouds) and atmospheric phenomena (due to its vicinity to the Sahara the effect of calima[1] must be taken into account). To study the effects of such phenomena and to identify affected data, a userfriendly webinterface is created. It provides an easy and performant tool to extract relevant information and create a data set as input for the user's analysis.

First, some general information about the astronomical background is given and the FACT project is introduced. The tables from the FACT database which are most important for the data check website are presented and all user requirements given by the FACT collaboration are described. Then the basic tools for creating the website (HTML, CSS, PHP and JavaScript) are displayed and code examples on how these were used are given. Next, a short introduction to the JavaScript library *jQuery* – which eases a lot of work when it comes to web programming – is presented. In order to send SQL statements to the database and retrieve the results, the website uses *Ajax Calls*. To display these query results, a datatable is the preferred tool. Here, *jQuery* offers a nice plugin, namely *jQuery DataTable*. The final component of the website comprises the plotting of the data, for which two different tools are provided: *Highcharts* and *jqPlot*. To prove the functionality of the website, examples have to be given. A data check is carried out with the help of the provided tools and typical influences on the data are pointed out. Last, the work is summarized and an outlook is given.

[1] weather phenomenon: intensely dry, warm and often dust-laden layer of the atmosphere which has its origin in the Sahara Desert. Due to strong winds it is driven out over the ocean and thus sometimes covers the Canary Islands.

Kurzbeschreibung

Das First G-APD Cherenkov Telescope (FACT) befindet sich auf der kanarischen Insel La Palma, Spanien. Es ist das erste Teleskop, welches spezielle Photosensoren, sogenannte Geiger-mode Avalanche Photodioden (G-APDs) benutzt, um aktive Galaxienkerne (AGN - active galactic nuclei) im TeV-Energiebereich zu beobachten. Da es ein erdgebundenes Teleskop ist, werden die Beobachtungsdaten be-einflusst, beispielsweise durch variierende Lichtbedingungen während einer Nacht (z.B. durch Mondlicht), durch Wetterbedingungen (z.B. Wolken) oder durch atmosphärische Phänomene (aufgrund der geografischen Lage muss vor allem in den Sommermonaten mit Calima[2] gerechnet werden). Um den Effekt dieser Wetterphänomene zu studieren und die betroffenen Daten herauszufiltern, wird ein benutzerfreundliches Webinterface programmiert. Es stellt ein leichtes und performantes Hilfsmittel dar und hilft beim Erstellen eines für die Analyse des Benutzers sinnvollen Datensatzes.

Zuerst wird eine kurze Einführung zum astronomischen Hintergrund gegeben und das FACT Projekt vorgestellt. Die wichtigsten Tabellen der Datenbank des Teleskops werden beschrieben und alle Benutzeranforderungen für die Website aufgestellt. Danach werden die Grundwerkzeuge beschrieben, mit denen die Website erstellt wurde (HTML, CSS, PHP und JavaScript), wozu jeweils Codebeispiele gegeben werden. Die JavaScript-Bibliothek *jQuery*, welche die Arbeit im Bereich der Webprogrammierung sehr erleichtert, wird vorgestellt. Um einen SQL-Befehl zu senden und das Ergebnis zu empfangen, werden *Ajax Calls* verwendet. Um die Ergebnisse der SQL-Abfragen darzustellen, wird eine Tabelle verwendet. Hier bietet *jQuery* ein sehr schönes Plugin an, welches das Erstellen dieser Tabelle sehr einfach macht: *jQuery DataTables*. Die letzte Komponente der Website ist das Tool zum Plotten der Daten, wobei hier dem Benutzer zwei Auswahlmöglichkeiten gegeben werden: *Highcharts* und *jqPlot*. Nachdem nun die Website erstellt und alle Funktionen eingebaut sind, werden Beispiele gezeigt, die die Funktionalität der Website beweisen. Ein Datencheck wird mit Hilfe der bereitgestellten Mittel durchgeführt und die häufigsten Störeinflüsse auf die Daten präsentiert. Zuletzt wird eine kurze Zusammenfassung verfasst und ein Ausblick auf Erweiterungsmöglichkeiten gegeben.

[2] Wetterphänomen: sehr trockene, warme und oft Sandstaub enthaltende Atmosphärenschicht, welche ihre Ursprünge in der Sahara hat. Aufgrund starker Winde wird sie auf das Meer hinausgetrieben und kann dadurch die Kanarischen Inseln erreichen.

Acknowledgements

This thesis has benefited greatly from the support of many people, some of whom I would like to thank here.

Firstly, I am sincerely grateful to Prof. Matthias Kadler for giving me the opportunity to combine my personal interest in astronomy and my course of studies by offering me this interesting topic.

Further I want to thank Dr. Daniela Dorner for her great support and guidance, for the countless tests of the website and the answering of the many questions concerning FACT and its astronomical background.

Many thanks to Prof. Hakan Kayal as the second corrector, without whose support this thesis at the Department for Astronomy would not have been possible.

I have to thank Lorenz Weber and Andre Löffler who shared their experiences as long time web programmers with me and gave me – as an absolute beginner – very useful advice about appropriate tools for such a data check website.

Many thanks to Miguel Claro for letting me use his fantastic picture titled “FACT Cherenkov Telescope in a Milky Way Backlight” on the cover of this thesis.

Contents

Abbreviations

AGN	Active Galactic Nuclei
Ajax	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
FACT	First G-APD Cherenkov Telescope
G-APD	Geiger-mode Avalanche Photodiodes
HTML	HyperText Markup Language
IACT	Imaging Atmospheric Cherenkov Telescope
JSON	JavaScript Object Notation
PDO	PHP Data Objects
PHP	PHP: Hypertext Preprocessor
SED	Spectral Energy Distribution
SQL	Structured Query Language
TeV	Tera Electron Volt

1 Introduction

1.1 Scientific Motivation

1.1.1 A Short Introduction to Active Galactic Nuclei

In astronomy, the term Active Galactic Nuclei (AGN) refers to central regions (the nuclei) of galaxies that show energetic phenomena that cannot be attributed directly to stars. To be classified as an AGN, a galaxy has to have a high luminosity (about 10^{11}-10^{14} times the luminosity of the sun), a small emitting region with extremely large energy densities, a supermassive black hole (about 10^8 solar masses) in the center and an emission range, that varies from radio over optical and X-ray even to gamma-ray bands.

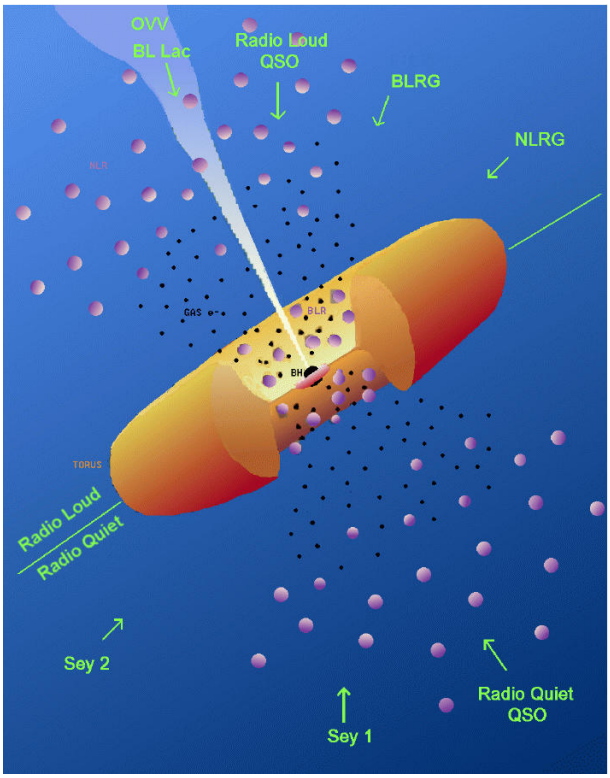


Figure 1.1: Illustration of the Unified Scheme of AGN as described in Urry and Padovani (1995). Different types of Active Galactic Nuclei show different characteristics depending on the angle under which they are observed. [1]

[1] Credits: http://www.cv.nrao.edu/course/astr534/ExtraGalactic.html

Classifying AGN is not very easy since various definitions can be applied. However, two main subclasses can be created: "radio-loud" and "radio-quiet" AGN. The group of radio-quiet AGN is formed by Seyfert galaxies and radio-quiet "quasars" (short for "quasi stellar object"). BL Lacertae objects, blazars, radio-loud quasars, optical violent variable quasars (OVV), broad and narrow line radio galaxies (BLRG resp. NLRG) are classified as radio-loud AGN. As can be seen in Figure 1.1, the AGN type depends on the angle that the jet axis has towards the Earth.

If this angle is very small, the jet points directly towards the Earth and we observe "down" the jet. These galaxies are the aforementioned blazars. What makes blazars special is the fact that their jets are moving close to the speed of light which indicates relativistic beaming. The maximum energies observed so far from their jet particles reach up to 20 TeV.

AGN are multiwavelength emitters which results in the need of observing them with a wide variety of telescopes. The characteristical plot for each of these galaxies is the so called "Spectral Energy Distribution" (SED). The SED shows the frequency on the x axis and the flux density on the y axis. For blazars, this SED has two peaks: one in the ratio to X-ray and one in the gamma-ray band (see Figure 1.2).

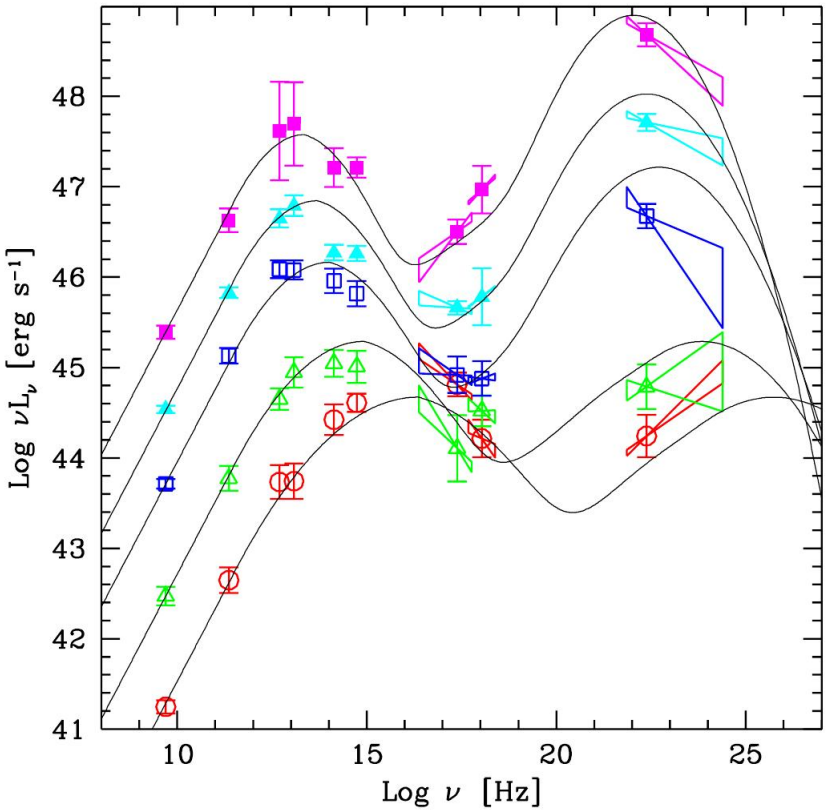


Figure 1.2: Sequence of average SEDs of blazars. The luminosity is plotted against the frequency. The continuous lines represent fits to the SEDs of different blazars. Credits: Donato et al., 2001 [5]

A branch of the field of AGN is the study of the variability of these galaxies. AGN show strong flux variations over the entire electromagnetic spectrum. Until today, the variations are mostly aperiodic and the amplitudes vary also. Thus it is very difficult to detect these variations. But with longterm monitoring of the sources and the help of so called "multiwavelength campaigns" (i.e. several telescopes monitor the source at different wavelengths at the same time), the variabilities can be recorded and correlated between different energies and thus contribute to the understanding of these phenomena and the physical processes behind it. While in the radio and optical range a lot of monitoring data is available, the data samples at TeV energies are rather sparse. The observations in this energy range are done from the ground with an indirect measurement technique. In order to monitor AGN at these energies, special telescopes are needed - so called "Cherenkov Telescopes". [18], [13]

1.2 The γ-ray telescope FACT

When γ-rays hit the Earth's atmosphere, particle cascades are triggered that generate Cherenkov light. This light can be detected by so called "Imaging Atmospheric Cherenkov Telescopes" (IACTs). One of these IACTs is the First G-APD Cherenkov Telescope (FACT) which took up its work in 2011. It is located on Roque de los Muchachos, La Palma, Spain and is the result of a collaboration of universities and institutes from Switzerland and Germany. FACT is observing cosmic gamma-ray sources at very high energies for example AGN. It is able to detect the Cherenkov photons on a nanosecond time scale when the generated cascades hit the Earth's surface. The camera records an image of the cascade. From the morphology and orientation of the shower image, the energy, type and direction of the primary particle can be reconstructed. The FACT camera has a field of view of 4.5 degrees and it consists of 1,440 G-APDs. These photodiodes are not ageing with strong light, so obervations during bright moon phases are possible. This enlarges the duty cycle of the instrument and also provides a stable performance, which is ideal for long term monitoring.

1.2.1 The Database

When FACT takes data, raw data and auxiliary files are recorded. These data are analysed and via mysql automatically stored in different tables in a database. Some of the most important tables are listed up here since not all contain relevant information for a data check:

- The **raw scientific data** are taken in so called "runs", where each run has a certain length (usually 1-5 minutes). Each raw data file consists of a header and

the data itself. The information about the system setup and configuration during the run and some event statistics are extracted from the header and stored in the table *RunInfo*. Also from the auxiliary files, information is retrieved on a run basis and filled into this table. Consequently, *RunInfo* contains all information available from the data taking.

- The table *ObservationTimes* contains information about the observation, for example begin and end of the astronomical twilight which are the **time limits for data taking**.

- Information about the **individual observed source** can be found in the table *Source*. Instead of always storing the name as a string, a unique key number (the SourceKey) is used to identify the object. This results in an optimization of the database since it saves memory and increases the access speed.

- There can be **various types of runs**, e.g. data or calibration runs, which are specified by a "run type key". Detailed information about the types of the keys is stored in the table *RunType*.

- If there were **problems or inconsistencies during data taking**, the person doing the shift can insert comments into the database, for example "high humidity during run". This information can be found in the table *RunComments*.

- Whenever a source is observed, FACT does not put the position of the source in the center of the camera but always switches ("wobbles") between two positions ("wobble positions") which are at an 0.6 degree offset from the camera center. The reason for this is very simple: it allows a simultaneous determination of the flux of background events. After a wobble position is changed, FACT does not only take data but also other runs that are needed for the calibration of the telescope. For calibration purposes, every 20 minutes a set of calibration runs is taken from which the background for signal extraction and calibration constants are extracted. Such a set of calibration and data runs is called a "sequence"[2]. To get information about the individual sequences the tables *SequenceComments* - where analoguely to *RunComments* comments on the sequence can be found - and *SequenceInfo* are available.

- There are **four analysis** tables (*AnalysisResultsNightLP*, *AnalysisResultsNightISDC*, *AnalysisResultsRunLP*, *AnalysisResultsRunISDC*) which contain the results of the analysis. As the name indicates, the analysis is done on a run and a nightly

[2] In addition, every 70 minutes a set of different calibration runs called "drs-sequence" is taken which is needed are needed to calibrate the DRS chip. To each data-sequence a drs-sequence is attributed.

basis. Furthermore, there are two analyses: a so called "quick look analysis" which is running on site in La Palma (LP) and an analysis which is running at the data center (ISDC) where the data are processed slightly different. For example, with each analysis software improvement a reprocessing of the complete data sample is performed in the data center.

To allow for a more flexible data check (rejection on run basis) and to use the best analysis, mainly the values from the tables *RunInfo* and *AnalysisResultsRunISDC* are used for the data selection.

1.3 User Requirements

As an earthbound telescope FACT has to deal with atmospheric influences on the data. The data can be affected by various night sky background light sources (e.g. the moon), by weather conditions (clouds, high humidity) or by weather phenomena typical for the Canary Islands like calima[3]. In order to help the user in finding affected data, a webinterface is created. This results in various requirements for the website, since a user wants to

- **plot parameters** that describe the data which results in requirements for the plotting tool:
  - Since some queries can return a lot of data points (up to 50,000), the tool must be able to plot many points in an adequate time.
  - Different types of plots must be possible, e.g. line, scatter or histogram.
  - The user must be able to zoom into the plot.
  - The tool must be able to display more than one y-axis.
  - An optional feature is the possibility to save the plot as an image file.
- **access and plot values** from the database without having to know the exact database structure and its content as well as SQL syntax.
  - The database is displayed in an intuitive structure and the user can easily interact with it.

[3] weather phenomenon: intensely dry, warm and often dust-laden layer of the atmosphere which has its origin in the Sahara Desert. Due to strong winds it is driven out over the ocean and thus sometimes covers the Canary Islands.

- **define aliases**:
  - Long terms do not need to be typed again, one simply has to define a short alias and can use this in other input fields.
- **interact easily** with the website:
  - Various buttons are needed, e.g. for submitting a query or plotting the data.
  - Since several y axes can be defined, every input field should have three buttons: an "add" button to add another input field, a "delete" button to remove the last added input field and a "reset" button to delete all inserted text.
  - If the user wants to set limits (e.g. "the value of the data points from this table must be bigger than two"), cuts can be defined.
- **see the query result** in a datatable that allows the (de-)selection of single data points to exclude them from the plot:
  - It is also possible to display additional columns in the datatable which should not be plotted.
  - The user can search the datatable for certain values.
  - Additionally, the table and the corresponding data can also be saved in a file if the user wants to export it to process it with another tool.

Whenever the database is changed (e.g. tables are added, removed or changed), the website automatically includes the new structure. Another important aspect is the security of the website. Since it has direct access to the FACT database, user access should be restricted to FACT members only. This can be achieved by integrating the FACT members' user accounts.

With the help of this website, the user can study the dependencies of parameters which helps in better understanding the telescope performance. The data quality of a chosen source can be studied by the user and a data set can be selected based on quality parameters. Thus it is possible to draw conclusions on the origin of the variability.

2 The Look: The Foundation Framework

When it comes to creating a website, one will inevitably have to deal with the most important tools for this, namely HyperText Markup Language (HTML), Cascading Style Sheets (CSS), PHP: Hypertext Preprocessor (PHP, recursive acronym) and JavaScript. Since a complete introduction to all these programming languages would go beyond the constraints of this thesis, I will only give a brief overview of each and show relevant examples from the created website.

2.1 HTML

As the name already implies HTML is a textbased markup language where a markup is defined by so called "tags". Tags are words enclosed in angle brackets and mostly come in pairs, namely a start and an end tag. End tags are marked with a slash before the word. Every website consists of three main elements: the *html* tag which surrounds the entire code, the *head* tag which contains information like the title, the used scripts or links to included files and the *body* tag which tells the browser what content to display. Thus a simple website – a blank page – has the following structure:

```
<html>
 <head>
 </head>
 <body>
 </body>
</html>
```

2.1: Header of website

Since HTML undergoes regular updates, different versions of it exist. So every website should tell the browser which version of HTML is to be used. Before HTML5 was released, this had to be done in the following way (example for HTML4):

```
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

I used the newest version HTML5 for the website which simplifies this tag to [19]:

```
<!doctype html>
```

For completeness and to achieve a good coding style, the encoding for the website should be specified. This is done with the “meta” tag.

```
<meta charset="utf-8">
```

Every code should contain comments for better understanding. In HTML <!-- marks the begin and --> the end of a comment.

```
<!-- this is a comment in HTML-->
```

So the basic HTML structure of a website with the title “FACT Data Check” looks like this:

```
<!doctype html>
 <html>
  <head>
   <title>FACT Data Check </title>
   <meta charset="utf-8">
   <!--furthermore, scripts and styles are defined here-->
  </head>
  <body>
  <!--here all displayed components are generated-->
  </body>
 </html>
```

2.2: Extended header of website

2.2 CSS

HTML without CSS is like a comic book without the pictures.
(Oliver Arscott) [1]

In order to improve userfriendliness and maintainability every modern website uses CSS. It allows you to define colors, layout, fonts and many other things. Since CSS comprises a lot of commands and positioning several elements in one single website can be very complicated, a friend of mine recommended to use a framework called *The ZURB Foundation Framework*[8] which eases the work of web programmers. It contains templates for many elements like buttons, typography or forms. One can easily compose a personal framework, download it and include its scripts and stylesheets in the website:

```
<!doctype html>
 <html>
  <head>
   <title>FACT Data Check </title>
   <meta charset="utf-8">
   <!--furthermore, scripts and styles are defined here-->
```

```
    <!--foundation framework and css sheets-->
    <script src="js/foundation.min.js"></script>
    <link type="text/css" rel="stylesheet" href="css/normalize.css" />
    <link type="text/css" rel="stylesheet" href="css/foundation.css" />
  </head>
  <body>
  <!--here all displayed components are generated-->
  </body>
</html>
```

2.3: Extended header of website

Explaining every single detail of Foundation would go beyond the scope of this thesis, so I will give a brief introduction to the most used components in the website.

A good HTML style is to put components into "containers" which are marked by the tag "div":

```
<div>
  <!--components can be defined in here, like buttons or textfields-->
  <input type="text" id="aliasTextfield">
  <input type="button" id="resetAll" value="reset all values">
</div>
```

When it comes to placing these elements (somewhere else than the default place), they have to be given different CSS values like margin, padding and position values. This can be complicated and time consuming. But here Foundation offers a very big relief: the so called "Foundation Grid". Foundation can divide the website up into rows. To create such a row one simply has to add a class attribute with the value "row" to the div element:

```
<div class="row">
  <!--components can be defined in here, like buttons or textfields-->
  <input type="text" id="aliasTextfield">
  <input type="button" id="resetAll" value="reset all values">
</div>
```

Every row can be divided up into 12 parts (columns) and thus maximum 12 elements can be placed there. If for example a textfield's width should be equal to the page width, there is no need to define a column and one simply has to type:

```
<div class="row">
  <input type="text" id="textFieldID">
</div>
```

If we want to add a button to this textfield, we should define some columns to make it look good. In the following example, a textfield is created which is six columns long and a button is placed behind it, such that it should look like Figure 2.1. The corresponding code for the input field and the button is written very easily with the help of Foundation:

```
<div class="row">
```

```
<div class="large-12 columns">
  <div class="large-3 columns">
    <p>Please enter some text here: </p>
  </div>
   <div class="large-6 columns">
    <input type="text">
   </div>
   <input type="button" value="submit">
  </div>
</div>
```

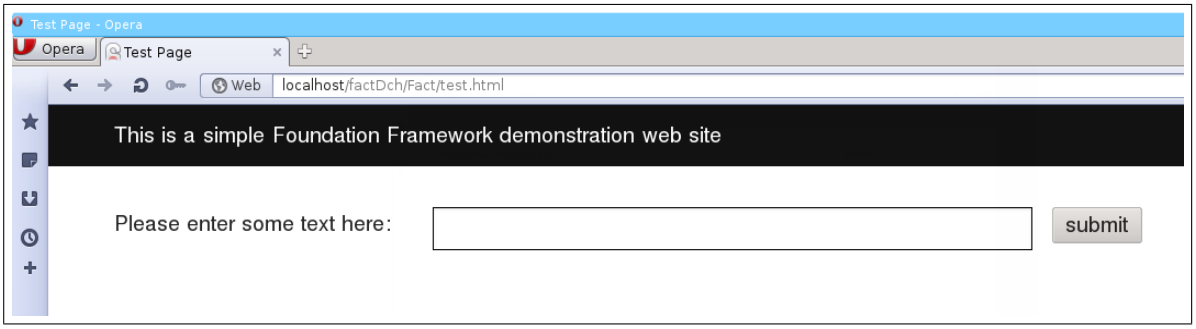


Figure 2.1: Screenshot of a demonstration website. A simple Foundation Framework example on how to use the "row" class is seen. The website contains an input field and a submit button.

It is also possible to nest the grid which enables the creation of complex layouts. The nesting is done intuitively by subdividing columns up again into columns:

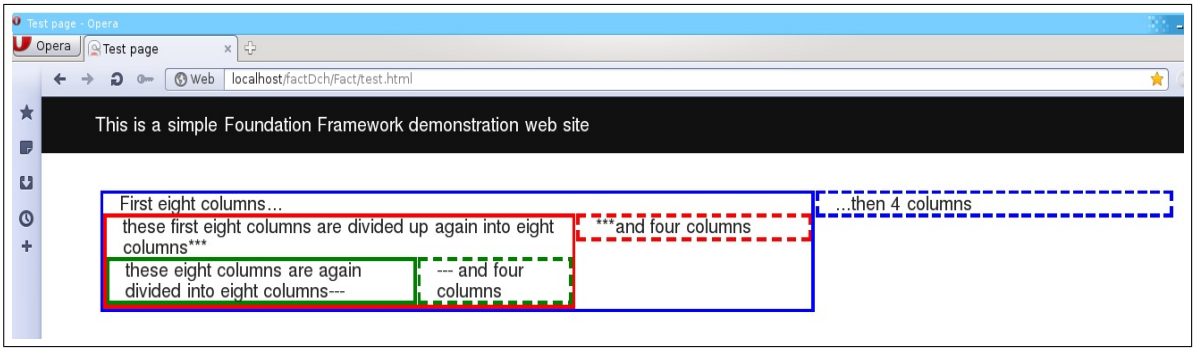


Figure 2.2: Screenshot of a demonstration website. A simple Foundation Framework example on how to nest columns is seen. First, a row is divided up into a block of eight and four columns (blue color), then the first block is subdivided again (red color), etc.

A third very nice feature is the help for so called "fieldsets". These are html input forms that can consist of several textfields and buttons. Mostly they are used for user inputs - a good example are shopping orders where name, address and phone number is to be inserted. Foundation makes it very easy to construct a fieldset with custom

textfields and buttons. The HTML code for a fieldset (see appendix, 1) leads to this result:

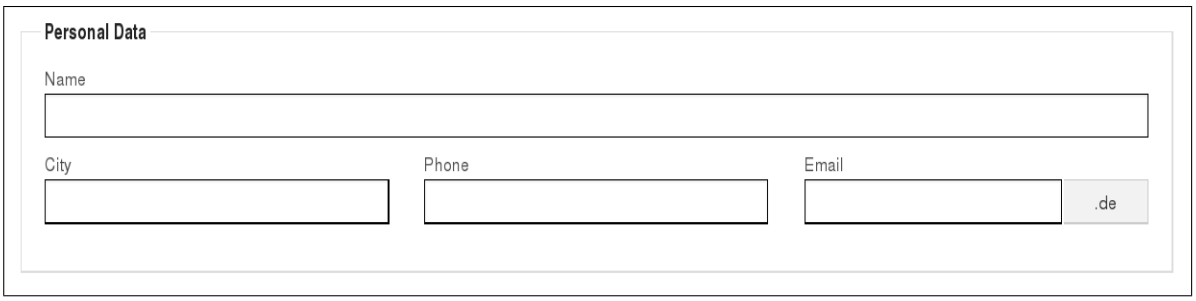


Figure 2.3: Screenshot of a demonstration website. A simple Foundation Framework example on how to create a custom fieldset is seen. The fieldset consists of input fields for name, city, phone number and email address.

With the help of these tools, it is already possible to create the HTML and CSS basis of the data check website.

2.2.1 The Elements of the Datatcheck Website

In order to make a data check possible, the website has to contain certain elements. Here are the main components:

1. the user must see the database and all its tables
2. there must be plotting tools for the data
3. the user must be able to select data which are displayed in a table
4. an SQL statement must be created from the user's input with which the database is queried. This requires various input fields, for example
   - define x and at least one y coordinate (since the data should be plotted)
   - define cuts for some values
   - define additional columns which are not plotted but only displayed in the table
   - define aliases to allow custom inputs for the user
5. various buttons for example for querying the database and plotting the data

With the help of the mentioned Foundation functions all required components are easily implemented in the website. The database element is located on the right side which eases the later explained drag and drop function (see chapter 3) for the user. On the left side, the user generated plot is always displayed at the top of the website, the various buttons and input fields are found in the center. The created SQL statement is also displayed in its own fieldset, right above the table containing the query data. The generation of SQL statement is done with the help of PHP.

Figures 2.4 and 2.5 show the look of the website after the user has been successfully logged in. The user name appears on the left side of the buttons.

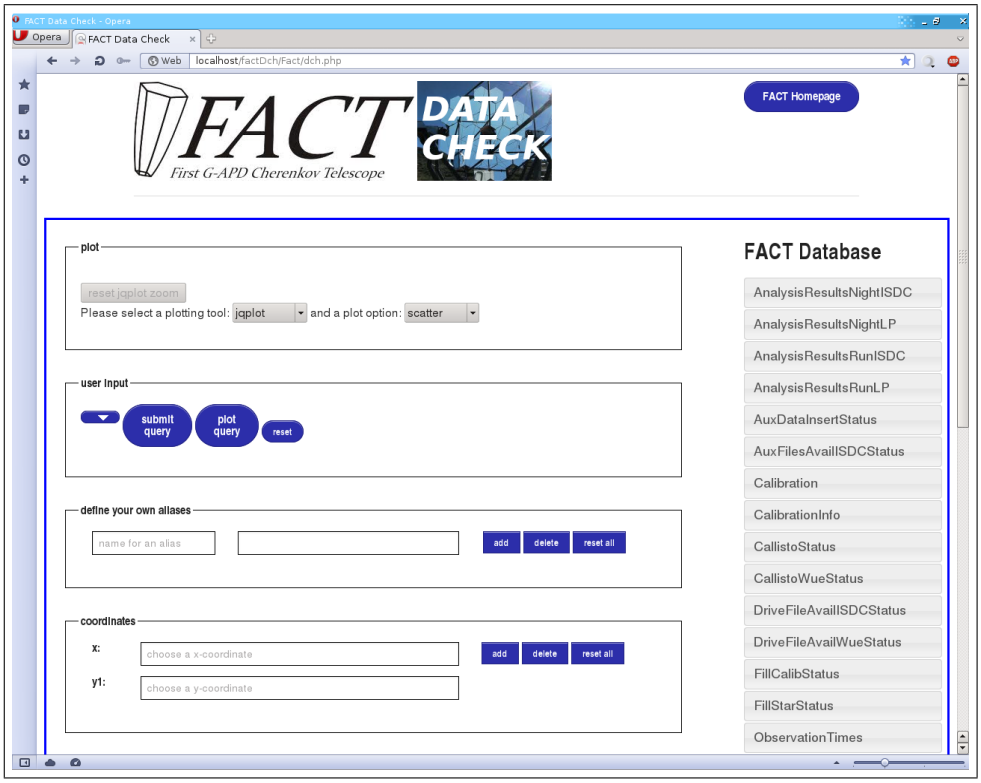


Figure 2.4: Screenshot of the data check website (upper part). The database is seen on the right side, the buttons and upper part of the input fields on the left.

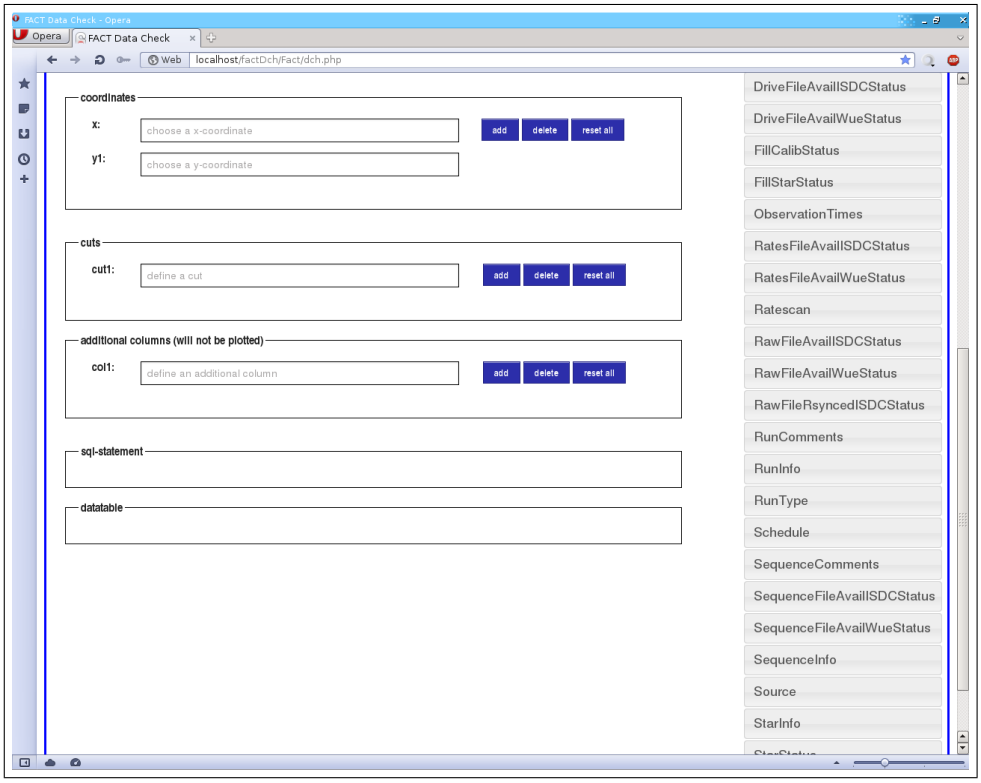


Figure 2.5: Screenshot of the data check website (lower part). The database is seen on the right side, the lower part of the input fields on the left.

2.3 PHP

PHP is a scripting language that is often used in web development as it can easily be embedded in HTML. One can simply write HTML and PHP code in the same file - provided that the file extension is *.php*. PHP code has to be written in between php tags:

```
<?php
// insert PHP code here
?>
```

PHP is - in contrast to JavaScript - a server-side scripting language and is interpreted by the web server. It is very comprehensive and a complete introduction would go beyond the scope of this thesis. Whenever I needed to look up something concerning PHP, I consulted the manual pages at `http://www.php.net/manual/de/index.php`. For the data check website I mostly worked with variables and functions:

```
// this is a variable declaration in PHP
// every variable has to begin with a dollar sign
$variableName = 5;
// this is a function in PHP; it is marked with the word function
function nameOfTheFunction() {
  // do something
}
// there can also be functions with parameters, for example variables
   or arrays
function nameOfTheFunction(array nameOfArray) {
  // do something with nameOfArray
}
```

In case of the datatcheck website PHP is used to establish the connection to the database, to send SQL statements and to deal with their results.

2.3.1 PHP and PDO

The relevant information for the connection to the FACT database is comprised in a file called “db.php” (2.4) which is included in the website file. It contains the host name, the user, the password and the name of the database.

```
<?php
 $db_host = "127.0.0.1";
 $db_user = "factweb";
 $db_password = "hashOfSecretPassword";
 $db_name = "factdata";
?>
```

2.4: File db.php

In order to connect to the database via PHP the interface PDO (PHP Data Objects) is used. The corresponding code takes the data from db.php and establishes the connection:

```
include("db.php");
$connstr = sprintf('mysql:dbname=%s;host=%s', $db_name, $db_host);
$pdo = new PDO($connstr, $db_user, $db_password);
```

Since the database structure can be changed, the website needs to be flexible and detect all tables. This is done with an SQL statement that needs the name of the database as an input. In PDO, the statement needs to be prepared first and then a value is bound to it, in this case *$db_name*:

```
$statement_get_table_names = $pdo->prepare('SELECT TABLE_NAME FROM
   information_schema.TABLES WHERE TABLE_SCHEMA=:db AND
   TABLE_TYPE=\'BASE TABLE\'');
$statement_get_table_names->bindValue('db', $db_name);
```

The following SQL statement provides all the corresponding columns for a certain table with the name *$tblname*:

```
$tblname = null;
$statement_get_table_columns = $pdo->prepare('SELECT TABLE_NAME,
   COLUMN_NAME FROM information_schema.COLUMNS WHERE TABLE_SCHEMA=:db
   AND TABLE_NAME=:tbl');
$statement_get_table_columns->bindValue('db', $db_name);
$statement_get_table_columns->bindParam('tbl', $tblname);
```

Until now, this code does nothing visible to the eye, these are just preparations. When it comes to creating the part of the website where the user can see the tables on the website, we simply need to execute the statements. But before that some HTML framework is needed. A very nice way to visually integrate the database is a "ToggleAccordion"[21]. The user can see the different tables (e.g. *AnalysisResultsNightLP*, *Calibration*) and by clicking on it, a list with all its columns is displayed. In Figure 2.6 the user can see that the table *AnalysisResultsNightLP* contains among others the columns *fNumEvtsAfterCleaning*, *fNumBgEvts* and *fOnTimeAfterCuts*.
The with PDO prepared SQL statements are executed by writing:

```
// this is a list of all tables of the database
$statement_get_table_names->execute();
```

To get all the corresponding columns, the PDO command *fetchColumn()* has to be executed in a while loop. In this loop, HTML code is generated that displays the table name and all columns are fetched.

```
// this is a list of all tables of the database
while ($tblname = $statement_get_table_names->fetchColumn()) {
 // print the name of the current table
 printf('<h4>%1$s</h4>', $tblname);
 // execute statement to get table columns
 $statement_get_table_columns->execute();
 // still a couple of things to do from here...
}
```

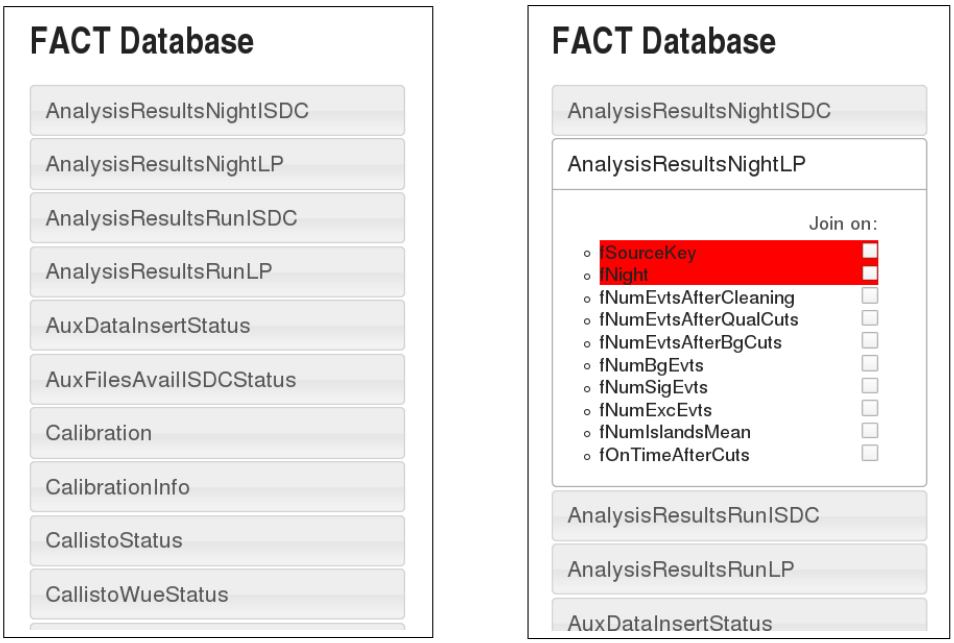


Figure 2.6: Look of the ToggleAccordion before (left) and after (right) the user clicked on a table name.

Additionally, I highlighted the primary keys in the table by assigning them with a red background color as can be seen in Figure 2.6. To find out which column is primary key, this statement has to be executed:

```
$keys = "SELECT k.column_name FROM information_schema.
    table_constraints t
   JOIN information_schema.key_column_usage k USING(table_schema,
      table_name)
   WHERE t.constraint_type='PRIMARY KEY' AND t.table_schema='$db_name'
       AND
   t.table_name='$tblname';";
// prepare and execute statement
$priKeyNames = $pdo->prepare($keys);
$priKeyNames->execute();
```

If one wants to put the result into an array via

```
$pri = $priKeyNames->fetchAll(PDO::FETCH_NUM);
```

the result yields an array in an array which can be flattened (i.e. a two dimensional array is turned into a one dimensional array) with the help of a PHP function (see appendix, 2). The final steps include generating HTML code with the *sprintf* function and adding checkboxes behind every column to give the user the opportunity to choose the columns to be joined over (see Fig. 2.6, right picture). The corresponding PHP code can be found in the appendix 3. Thus the user has the possibility to view the database and its columns, but so far nothing can be done with it. This is where JavaScript comes in.

2.4 JavaScript

JavaScript is a client-side scripting language that is used in web programming, and in the case of the data check website helps with user interactions. It reacts to user inputs like pressing buttons or filling out textfields. The most used characteristics for the website are variables and functions:

```
// line comments are generated like this
/* this is a comment comprising
   several lines */
// this is a variable in JavaScript
var a = 5;
// this is a function declaration
function nameOfTheFunction() {
 // do something here
}
 // this is a function with parameters
function nameOfTheFunction(para1, para2) {
 // do something here with para1 and para2
}
```

A big benefit of JavaScript is its ability to access HTML elements. This is done with the help of the Document Object Model (DOM) which is a convention for representing and interacting with objects in markup languages like HTML. An HTML page can be seen as a document which is organized in a tree structure with the topmost node as the "document object". A simple example for JavaScript and the usage of DOM is the calculation of the square of a number:

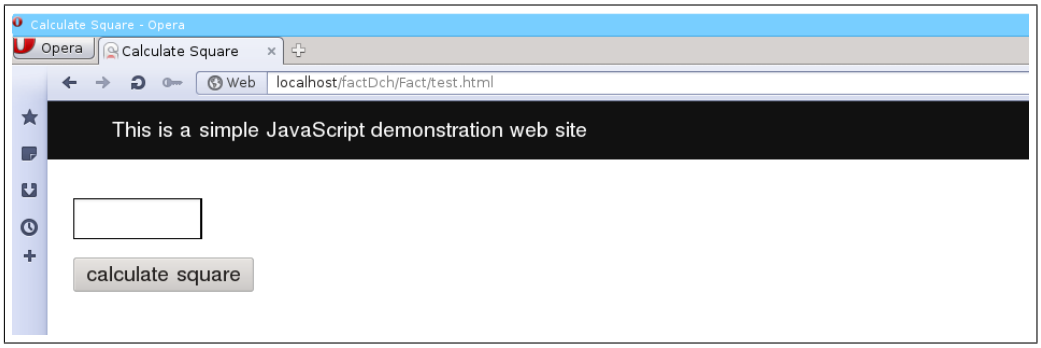


Figure 2.7: Screenshot of a demonstration website. A simple JavaScript example on a website that can calculate the square of a number inserted in the input field.

The HTML code creating this example will not react to any user input (i.e. pressing the button):

```
<html>
 <head>
  <title>Calculate square</title>
 </head>
```

```
 <body>
  <div class="top-bar"><p style="color: white; position: absolute;
    left: 50px; top: 10px;">
    This is a simple JavaScript demonstration web site</p>
  </div>
  <div class="row" align="left">
   <form name="formForSquare" action="">
    <input type="text" name="input" style="width: 200px;">
    <input type="button" value="calculate square">
   </form>
  </div>
 </body>
</html>
```

But with the help of JavaScript a function is assigned which will be executed on clicking:

```
<input type="button" value="calculate square" onclick="calc()">
```

The JavaScript code has to be written in between "script" tags with the type specified as JavaScript. The function *calc()* has to be defined here:

```
<script type="text/javascript">
 function calc() {
  var result = document.formForSquare.input.value *
     document.formForSquare.input.value;
  alert("The square of " + document.formForSquare.input.value +
     " is " + result);
 }
</script>
```

When the button is pressed, JavaScript generates a pop up window ("alert") that prints the result:

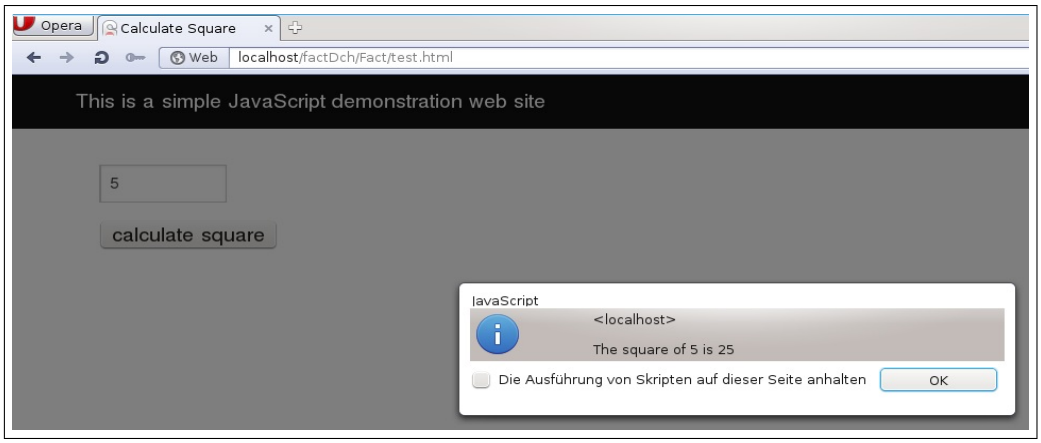


Figure 2.8: Screenshot of a demonstration website. Result of the calculation of the square with the JavaScript example.

This is only a very simple example, and it is obvious that more difficult websites lead to more complex JavaScript code. Especially long terms like

```
document.getElementById.input.value
```

become more complicated and confusing. To avoid this, jQuery is used.

3 Simplifying JavaScript: jQuery

jQuery [10] is pure JavaScript – in fact it is a JavaScript library. It was released in 2006 but is already one of the most popular libraries. Including it in a website is very easy, I just added a script tag in the head of the file dch.php. In this case, the code for jQuery comes straight from the developers' website which means that the files do not have to be downloaded. It is important to include jQueryUI, too, since this is in charge of interface interactions and effects:

```
<!doctype html>
<html>
 <head>
  <title>FACT Data Check </title>
  <meta charset="utf-8">
  <!--furthermore, scripts and styles are defined here-->
  <!--include jquery directly via internet-->
  <script type="text/javascript"
    src="http://code.jquery.com/jquery-1.10.1.js"></script>
  <script type="text/javascript"
    src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
  <!--foundation framework and css sheets-->
  <script src="js/foundation.min.js"></script>
  <link type="text/css" rel="stylesheet" href="css/normalize.css" />
  <link type="text/css" rel="stylesheet" href="css/foundation.css" />
 </head>
 <body>
  <!--here all displayed components are generated-->
 </body>
</html>
```

3.1: Extended header of website

jQuery was developed to ease the use of JavaScript code. For example, if there is a button on a website and in case someone clicks on it, a CSS class (which is used to specify a style for a group of elements) will be added to it. If it is clicked again, this class should be removed. This procedure is called "toggling". In JavaScript, this has to be done like this:

```
var button = document.getElementById('submitButton'),

 hasClass = function (el, cl) {
  var regex = new RegExp('(?:\s|^)' + cl + '(?:\s|$)');
  return !!el.className.match(regex);
```

```
 },

 addClass = function (el, cl) {
  el.className += ' ' + cl;
 },

 removeClass = function (el, cl) {
  var regex = new RegExp('(?:\s|^)' + cl + '(?:\s|$)');
  el.className = el.className.replace(regex, ' ');
 },

 toggleClass = function (el, cl) {
  hasClass(el, cl) ? removeClass(el, cl) : addClass(el, cl);
 };

toggleClass(button, 'anotherClass');
```

And here is what has to be done in jQuery:

```
$('#submitButton').toggleClass('anotherClass');
```

That is 16 lines of code in JavaScript and one line in jQuery. Thus using jQuery means that the web programmer does not have to write much code which results in less time to write code and easier finding of errors. Additionally, jQuery is very intuitive. I will give a short overview of some features of jQuery since I used it very often for the data check website.

In order to use jQuery, the jQuery object has to be called. This is done either by using the dollar sign or by directly writing "jQuery":

```
// long version
jQuery('#buttonID')
// short version
$('#buttonID')
```

Simple HTML elements like *div* or *h3* can be accessed with their label but since there may be more than one of these elements in the website, one can also access classes and IDs:

```
// accessing simple HTML elements
$(div)
$(h3)
// classes can be selected with a dot
$('.classOfDiv')
// IDs can be selected with a hash
$('#idOfDiv')
```

Variables and functions in jQuery are defined as in JavaScript. There exist a lot of built-in functions that make web programming a lot easier. Often HTML elements should be appended or prepended to existing elements. For example if a user wants to define more y coordinates for the plot, he/she presses the add button and then a new

input line is added. Analoguely it has to be removed if the user presses the delete button. The jQuery code looks like this:

```
// if you click on the "add a y coordinate"-button
$(".addY").click(function() {
 // call this function
 addButtonPressed("y", yArray);
});
// if you click on the "delete a y coordinate"-button
$(".delY").click(function() {
 // call this function
 deleteButtonPressed("y", yArray);
});

// this is only an extract of the full function
function addButtonPressed(type, array) {
  // ...
  } else if (type === "y") {
    // ...
    // append something to the fieldset for the coordinates
    $('#yBox').append("<div class=\"row\" id=... .... </div>");
  }
  //...
}
// if a delete button is pressed
function deleteButtonPressed(name, array) {
 // id of the last entry
 var currentID = array.pop();
 // remove the html element
 $("#" + name + "Box #" + currentID).remove();
 // ...
}
```

3.2: Add and delete button is pressed

Figures 3.1 and 3.2 show examples how the add and delete buttons work.

To spare the user the typing of each table and column name into the input fields, the column names are made draggable (one can click on their names and drag them around) and the input fields are made droppable (the dragged column names can be dropped in there). With the help of jQuery, this is done with a few lines of code. jQuery can bind event types to a chosen html part – in this case to the list item of the ToggleAccordion. The column names are bound to the event "mousedown" which means pressing the mouse and keeping it pressed for some time. Whenever this mousedown event happens, the following function is executed:

```
$("#toggleAccordion .selectable li").bind("mousedown", function(e) {
 e.metaKey = true;
}).draggable({

});
```

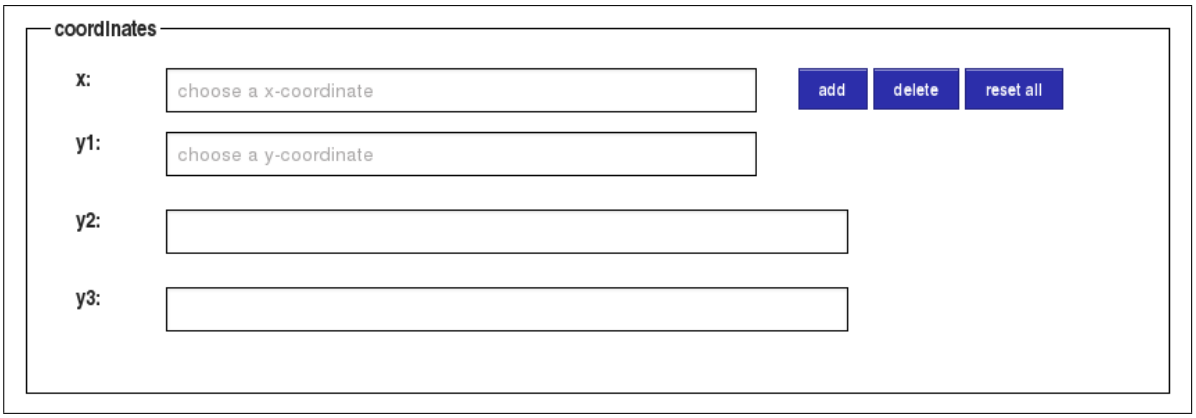


Figure 3.1: Screenshot of the data check website. One can see the fieldset in which the input fields for x and y axes are embedded. On the right, the buttons for adding, deleting and reseting the input fields are displayed. One can see that some additional input fields were defined with the help of the add button.

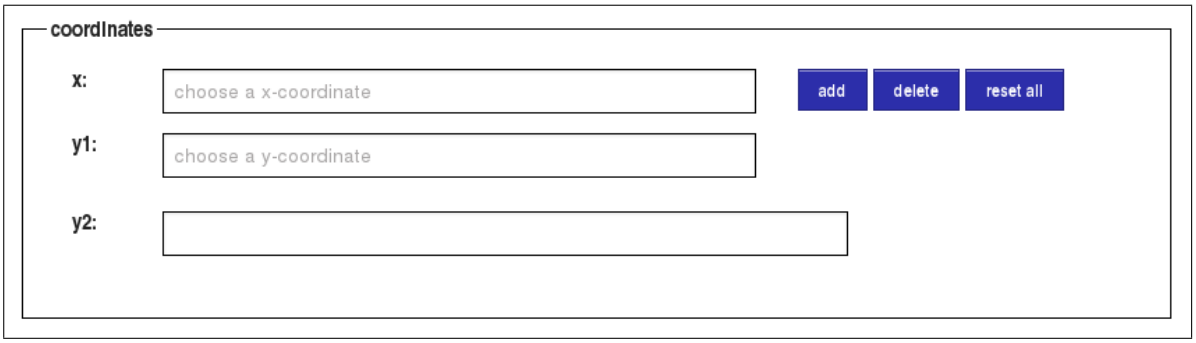


Figure 3.2: Screenshot of the data check website. One can see the fieldset in which the input fields for cuts an additional columns axes are embedded. On the right, the buttons for adding, deleting and resetting the input fields are displayed. Compared to Figure 3.1, one input field was removed with the delete button.

Now each list item in the ToggleAccordion – which is a certain column name – reacts to the mousedown event and is made draggable. However, if the user now clicks on the name and drags it out of the ToggleAccordion, the name would disappear from it. It is necessary to clone the name:

```
$("#toggleAccordion .selectable li").bind("mousedown", function(e) {
 e.metaKey = true;
}).draggable({
 helper: "clone"
});
```

Unfortunately, a problem occured after adding the "Join-on"-checkboxes after every column name: they were also cloned and visible when dragged. I fixed this by defining my own helper function. The column name itself is obtained by the *text()*-method, and

thus the helper only has to return a html element (a paragraph <*p*>) with the text:

```
$("#toggleAccordion .selectable li").bind("mousedown", function(e) {
 e.metaKey = true;
}).draggable({
 helper: function () {
  return $("<p></p>").append($(this).text());
 },
});
```

Since the database consists of many tables, it is possible that one has to scroll that far down that the input fields can not be seen anymore. When a column name is chosen the user has to scroll up manually while holding the chosen column name. To relieve the user from this, I defined a point where the website scrolls to automatically when the chosen name is dragged out of the ToggleAccordion area. This is done with the help of the start and stop function for the draggable function and the "mouseleave" event.

```
$("#toggleAccordion .selectable li").bind("mousedown", function(e) {
 e.metaKey = true;
}).draggable({
 helper: function () {
  return $("<p></p>").append($(this).text());
 },
 // if the mouse leaves the list item, scroll up to the coordinates box
 start: function() {
   $(this).bind('mouseleave', function() {
   $.scrollTo('#aliasBox', {duration: 'slow'});
  });
 },
 // make sure to unbind the mouseleave again
 stop: function() {
  $(this).unbind('mouseleave');
 }
});
```

3.3: Mouseleave event in ToggleAccordion

The jQuery scrollTo function is not a standard jQuery function and comes from a jQuery plugin [7] which is included in the head of the data check website.

```
<!--scrollTo plugin for jquery-->
<script type="text/javascript" src="js/jquery.scrollTo-1.4.3.1-min.js">
   </script>
```

Once a column name is chosen and dragged to an input field, it can be dropped in there. Again jQuery eases many things by providing a "droppable" function. Since there are many columns in the database which share the same name (e.g. fNight, fSourceKey) it is important to specify the corresponding table in the input field. Additionally, the dragged column name should be inserted at the position of the cursor in the input field. Again, jQuery offers a little plugin called "caret" [20] which determines the current

position of the cursor. All these things are done by the function “drop” which is called when the column name is dropped into an input field (which has the class “dropTarget”):

```
$(".dropTarget").droppable({
 drop: function(event, ui) {
  // name of the source table
  var tableName = ui.draggable.parent().attr('data-tablename');
  // chosen object value
  var target = $(this);
  // helper
  var helper = ui.helper;
  // current position of the cursor
  var pos = target.caret();
  // new value of the textfield
  var newVal = target.val().substring(0, pos) + tableName + "." +
      helper.text() + target.val().substring(pos);
  // set value
  target.val(newVal);
 }
});
```

3.4: jQuery function droppable()

With the help of this code, the user can easily fill out the input fields, as can be seen in Figures 3.3 and 3.4.

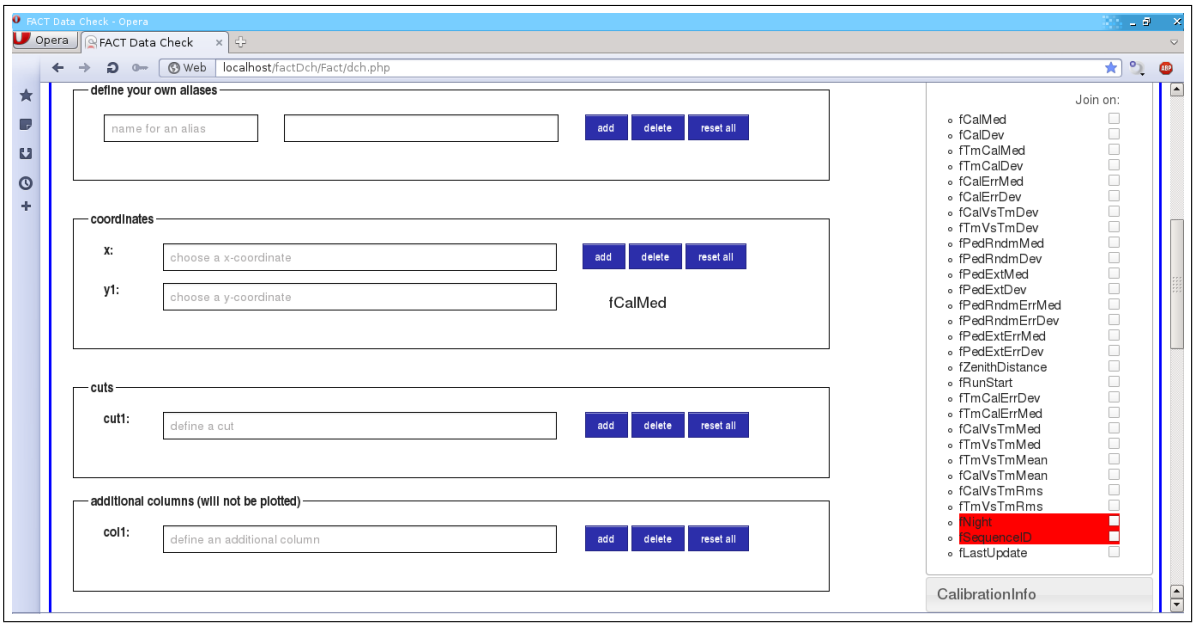


Figure 3.3: Screenshot of the website. The column name fCalMed is dragged from the table Calibration into an input field (the mouse can not be seen).

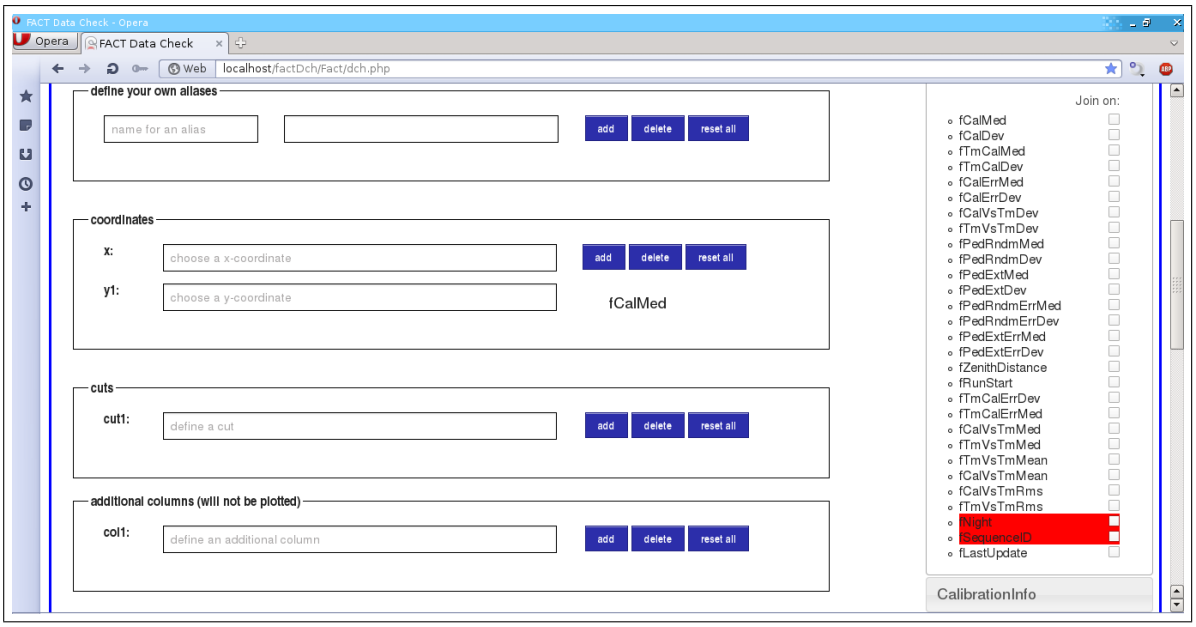


Figure 3.4: Screenshot of the website. The column name fCalMed has been dropped into the input field for the x axis. One can see that the table name is automatically inserted before the column name.

When the user has chosen values for the input fields – at least the x- and one y-coordinate has to be defined, otherwise an alert window informs the user about the missing value – the submit button can be pressed to create the SQL statement and to show the result in a datatable. In order to show the created SQL statement, a form is defined which at first contains only an empty div-element:

```
<form>
 <fieldset style="margin-right: -65px;">
  <legend>sql-statement</legend>
  <div id="statement"> <div>
 </fieldset>
</form>
```

The creation of this statement is done by the function *submit()*. Since the code would be too much to print here, I will shortly describe what it does:

- detect all values from the input fields
- replace aliases if the user defined any
- create the SELECT and WHERE part of the statement
- define the JOIN part of the statement
- send the statement to the database

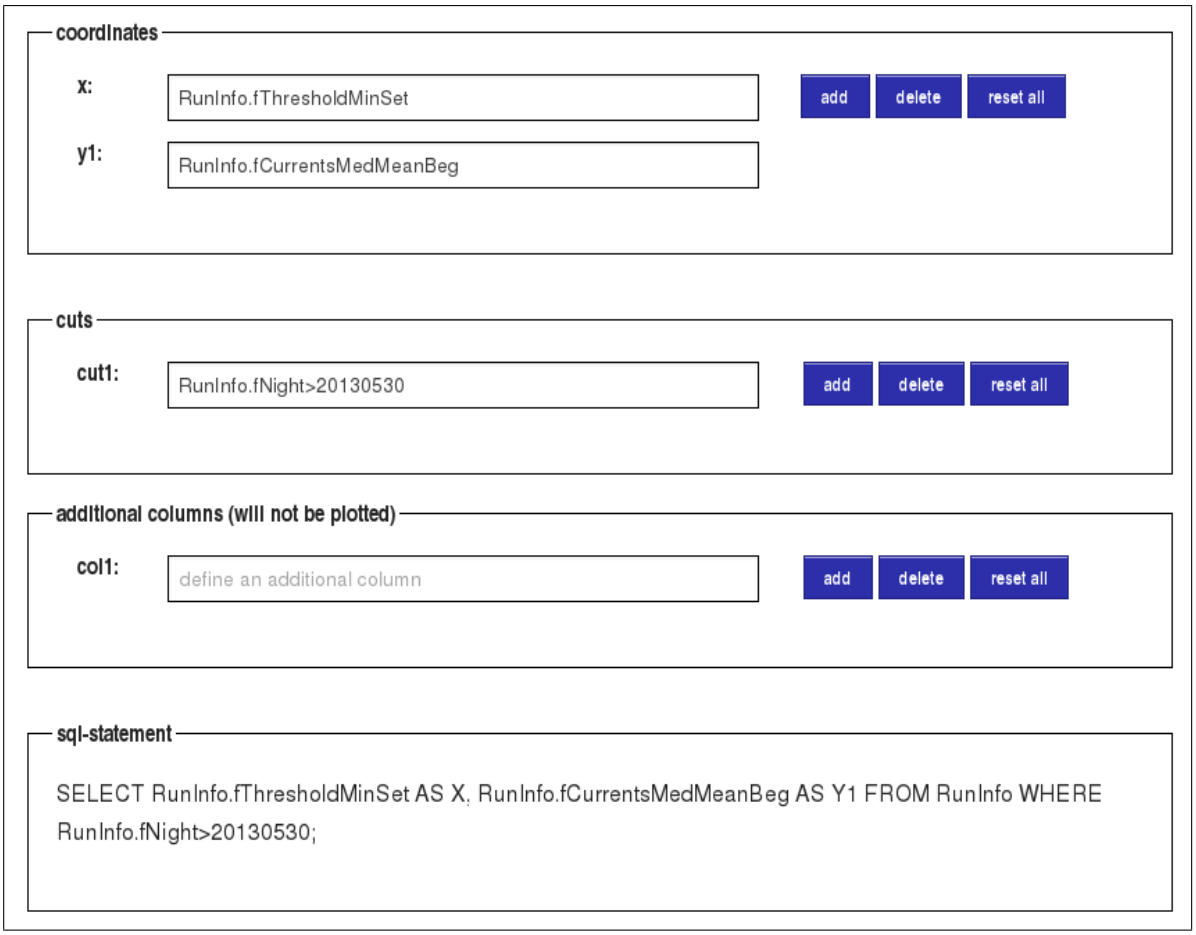


Figure 3.5: Screenshot of the website. The axes input fields and one cut input field have been filled out and the submit button was pressed. The generated SQL statement can be seen under the input fields.

When the submit button is pressed, the SQL statement string is appended to the div (see Figure 3.5):

```
$('#statement').append('<p>' + sqlStatement + '</p>');
```

The sending of the SQL statement requires a so called "Ajax call" (Ajax = Asynchronous JavaScript and XML) and some help from PHP.

4 Querying the Database: Ajax Calls

When the function *submit()* has generated the SQL statement as a string, it has to query the database with it. The problem is that the database can only be accessed with a server-side scripting language like PHP, but as mentioned before, jQuery works on the client-side. Fortunately, jQuery provides a very nice solution to this solve this issue. The sending of the statement and the retrieving of the result of the query is accomplished by so called "Ajax calls". jQuery offers the function *$.ajax()* which can send data to a php script, have it executed and returns the result. Thus a SQL query with jQuery works like this:

```
function ajax(sqlStatement) {
 // ajax function
 $.ajax({
    type: "POST",
    url: "sql.php",
    data: "sqlState=" + sqlStatement,
    datatype: "json",
    success: function(newData) {
    // here the returned data (newData) can be processed further
    };
 });
};
```

4.1: Function ajax()

The *$.ajax()* function itself has certain parameters like

- *type*: describes the method of passing arguments
- *url*: the file the statement is send to
- *data*: the data to send
- *datatype*: the datatype of the **returned** data; here JavaScript Object Notation (JSON) was chosen since it is a format that transmits data objects consisting of attribute-value pairs
- *success*: in case of a successful Ajax call, this function defines what to do with the returned data

The file sql.php (4.2) retrieves the sent data with the POST method and stores it in form of the variable *$name*. Then it is passed on to PDO, prepared and executed. Since the query result is always a form of data pairs, the JSON format provides a perfect structure for returning it and additionally, jQuery can easily handle JSON, too. To generate a JSON format the built-in PHP function *json_encode()* is used and sent back to the client-side with a simple "echo" command.

```
<?php
 include("db.php");
 $connstr = sprintf('mysql:dbname=%s;host=%s', $db_name, $db_host);
 $pdo = new PDO($connstr, $db_user, $db_password);

 $name = $_POST['sqlState'];
 $sqlStatement = $pdo->prepare($name);
 $sqlStatement->execute();
 $json = $sqlStatement->fetchAll(PDO::FETCH_NUM);
 echo json_encode($json);
?>
```

4.2: File sql.php

This file can be used not only for user generated but also for "internally" generated SQL statements. A good example is the generation of the JOIN part of the statement. There are two possibilities: the user chooses the columns to join over with the help of the checkboxes or the website defines it. In the latter case the involved tables and the corresponding columns need to be found and compared. If a column name appears in all of these tables, it can be used in the JOIN part. To get all columns of one table, an Ajax call needs to be executed (4.3). Since this function is called in a loop that goes through all relevant tables, it is important to avoid synchronous execution of the ajax command and to wait until the result is returned for each table. Otherwise it might happen that the Ajax call is not yet finished with the determination of all columns for one table but already starts the query for another table which leads to errors. To make sure that this does not happen, an additional parameter *async* has to be defined and set to "false":

```
// get all column names from the given table name
function getAllColumns(tableName) {
 var string = "SELECT COLUMN_NAME FROM information_schema.COLUMNS WHERE
     TABLE_SCHEMA='factdata' AND TABLE_NAME='" + tableName + "';";
 // return variable, Ajax call returns an array
 var dataArray;
 $.ajax({
  type: "POST",
  url: "sql.php",
  data: "sqlState=" + string,
  datatype: "json",
  async: false,
  success: function(newData) {
   dataArray = $.parseJSON(newData);
  }
```

```
 });
 return dataArray;
}
```

4.3: Function getAllColumns()

If a statement is returned from sql.php – meaning the query could be executed – the jQuery function *$.ajax()* takes the JSON object and processes it further with the help of *success*. Here a function is defined that takes the JSON object as a parameter. In order to work with this object it needs to be parsed by jQuery with the built-in function *parseJSON()*:

```
 success: function(newData) {
   dataArray = $.parseJSON(newData);
  }
```

When the result from the user-generated SQL statement is returned successfully from 4.2, the first thing to do is to check whether the JSON object is empty. This is the case, if the query returned an empty data set which happens because of an incorrect syntax or the query itself returns no data. This event triggers an alert window with a message for the user. If the data set is not empty, a datatable is created and displayed on the website.

```
function ajax(sqlStatement) {
 // ajax function
 $.ajax({
    type: "POST",
    url: "sql.php",
    data: "sqlState=" + sqlStatement,
    datatype: "json",
    success: function(newData) {
     if (newData === "[]") {
      alert("sorry, your request returned an empty set, please try
          again!");
      // ...
     } else {
      // show the result in a datatable with jQuery DataTables
     }
    };
 });
};
```

4.4: Extended function ajax() (1)

From this point on, another jQuery plugin takes over: jQuery DataTables.

5 The Datatable Plugin: jQuery DataTables

When the query is executed and the data are retrieved, they need to be displayed for the user in a datatable. Thanks to jQuery this task is done easily by integrating the plugin "jQuery DataTables" [12] and its method *dataTable()*. But before the table is displayed on the website and this method is called, some preparations are needed.

First of all, one needs to append a simple html table structure (<*thead*>, <*tr*>, <*th*>, <*table*>) to the div the datatable is shown in. The creation of the header (<*thead*>) and the rows (<*tr*>) is done with the function *createTableHeader()*. Second, a check is necessary if there is already a datatable displayed on the website (because the user has submitted a query some time earlier). If this is the case, it is removed from the div. Third, the html table specifications like cellpadding, cellspacing, id and class are directly defined in the *success* function:

```
$('#tableDiv').html('<table cellpadding="0" cellspacing="0" border="0"
    class="display" id="data"></table>');
```

After these steps are done, the created (and empty) html table is appended to the predefined div with the id "tableDiv" and the created table header is appended to the table with the help of the id "data". So the ajax function from 4.4 is extended to the following code:

```
function ajax(sqlStatement) {
 // create the html header string for the table
 var cols = createTableHeader();
 // in case the table was defined by a previous query, remove it
 $('#tableDiv').remove('#data');
 // ajax function
 $.ajax({
  type: "POST",
  url: "sql.php",
  data: "sqlState=" + sqlStatement,
  datatype: "json",
  success: function(newData) {
  if (newData === "[]") {
   alert("sorry, your request returned an empty set, please try again!"
       );
  } else {
   // append an empty html table to div
```

```
    $('#tableDiv').html('<table cellpadding="0" cellspacing="0" border
       ="0" class="display" id="data"></table>');
    // append created table header
    $('#data').html(cols);
    // create a datatable with jQuery dataTable
    table = $('#data').dataTable({
     // here the parameters of the datatable are defined
    });
   }
 });
}
```

5.1: Extended function ajax() (2)

jQuery DataTables offers many parameters that can be set and adjusted. The most important one of these is *aaData* which defines the data that the table should display. The other parameters are:

- *aLengthMenu*: menu in which allows the user to choose the number of table rows that are displayed.
- *iDisplayLength*: defines the number of rows that are shown when the table is generated for the first time.
- *sPaginationType*: defines the pagination interaction mode. Per default, only the "prev" and "next" button are shown but in case of the website, the page numbers are displayed.
- *oTableTools*: displaying of various buttons that provide options like exporting and saving the table or selecting rows.
- *sDom*: defines the location of the various controls of the datatable (e.g. pagination, row selection, search field).

So the final function *dataTable()* from function 5.1 looks like this:

```
table = $('#data').dataTable({
 "aaData": $.parseJSON(newData),
 "aLengthMenu": [[10, 20, 50, 100, 200, 500, 1000, 2000, -1], [10, 20,
    50, 100, 200, 500, 1000, 2000, "All"]],
 "iDisplayLength": 20,
 "sPaginationType": "full_numbers",
 "oTableTools": {
  "sSwfPath": "TableTools/media/swf/copy_csv_xls_pdf.swf",
  "sRowSelect": "multi",
  "aButtons": ["pdf", "csv", "select_all", "select_none"]
 },
 "sDom": 'T<"top"l>irt<"bottom"fp>'
});
```

5.2: jQuery DataTable parameters

When the table is fully generated, the website directs the user to it by scrolling automatically to its div with the id "tableDiv" by executing the jQuery *scrollTo()* function.

```
SELECT RunInfo.fZenithDistanceMean AS X, RunInfo.fThresholdMinSet AS Y1 FROM
RunInfo WHERE RunInfo.fSourceKEY=5 AND RunInfo.fThresholdMinSet IS NOT NULL
AND RunInfo.fNight>20130401 AND RunInfo.fZenithDistanceMean IS NOT NULL;
```

Figure 5.1: Screenshot of the website. An SQL statement with the x and one y axis and four cuts is defined.

The final datatable for the SQL statement in Figure 5.1 can be seen in Figure 5.2.

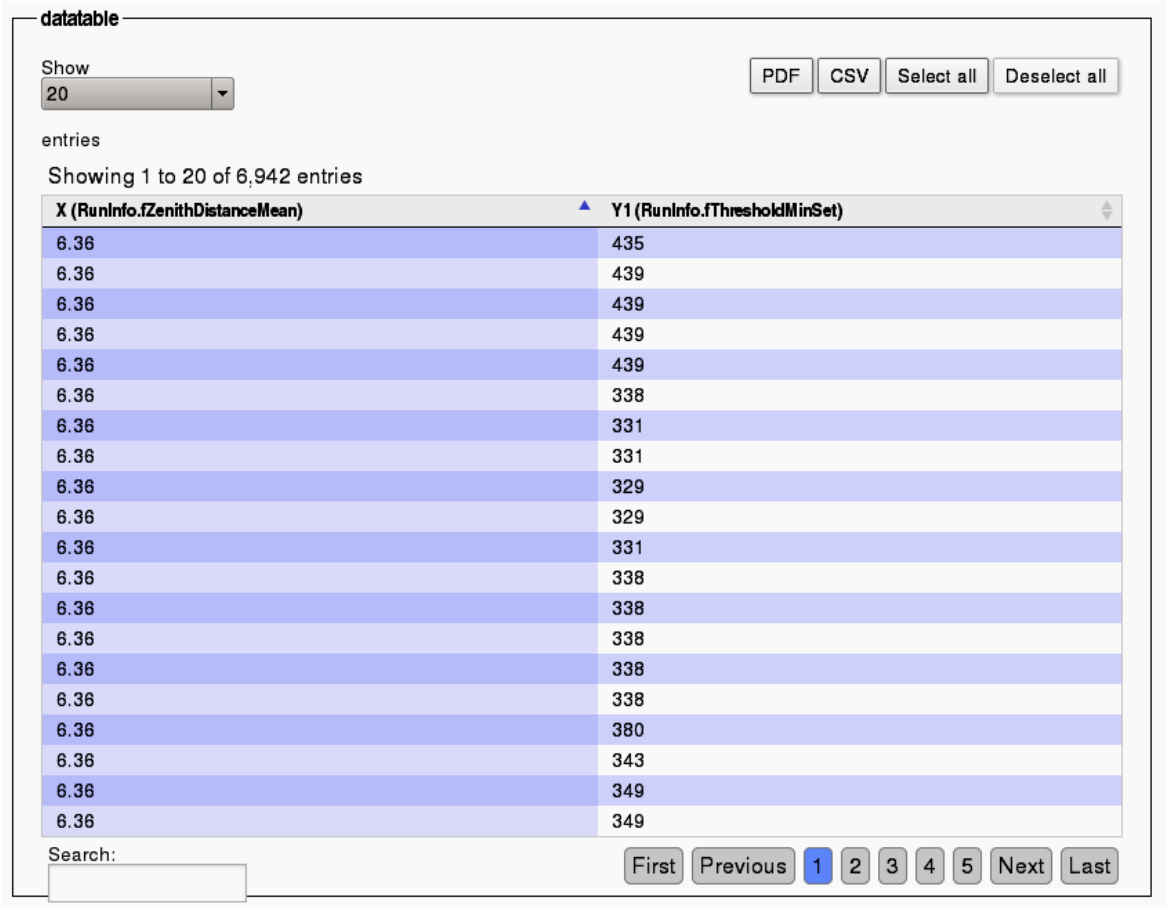


Figure 5.2: Screenshot of the website. The generated datatable from statement 5.1 is displayed. On the left upper side, the user can choose how many rows should be displayed. On the right upper side, the buttons for saving the data in a file and for selecting and deselecting all rows can be seen. The table itself is placed in the center, the name of the input fields are used as headers. Every single row is (de-)selectable. On the bottom left side, a search fields helps in finding values. On the bottom right side, the pagination buttons are placed.

In order to plot data, the user simply has to select all the rows he or she wants to plot. In order to select all rows, the datatable offers the button "Select All". Analoguely, "Deselect All" deselects all rows again. It is also possible to select and deselect rows manually by simply clicking on them. A selected row is marked with a darker background color:

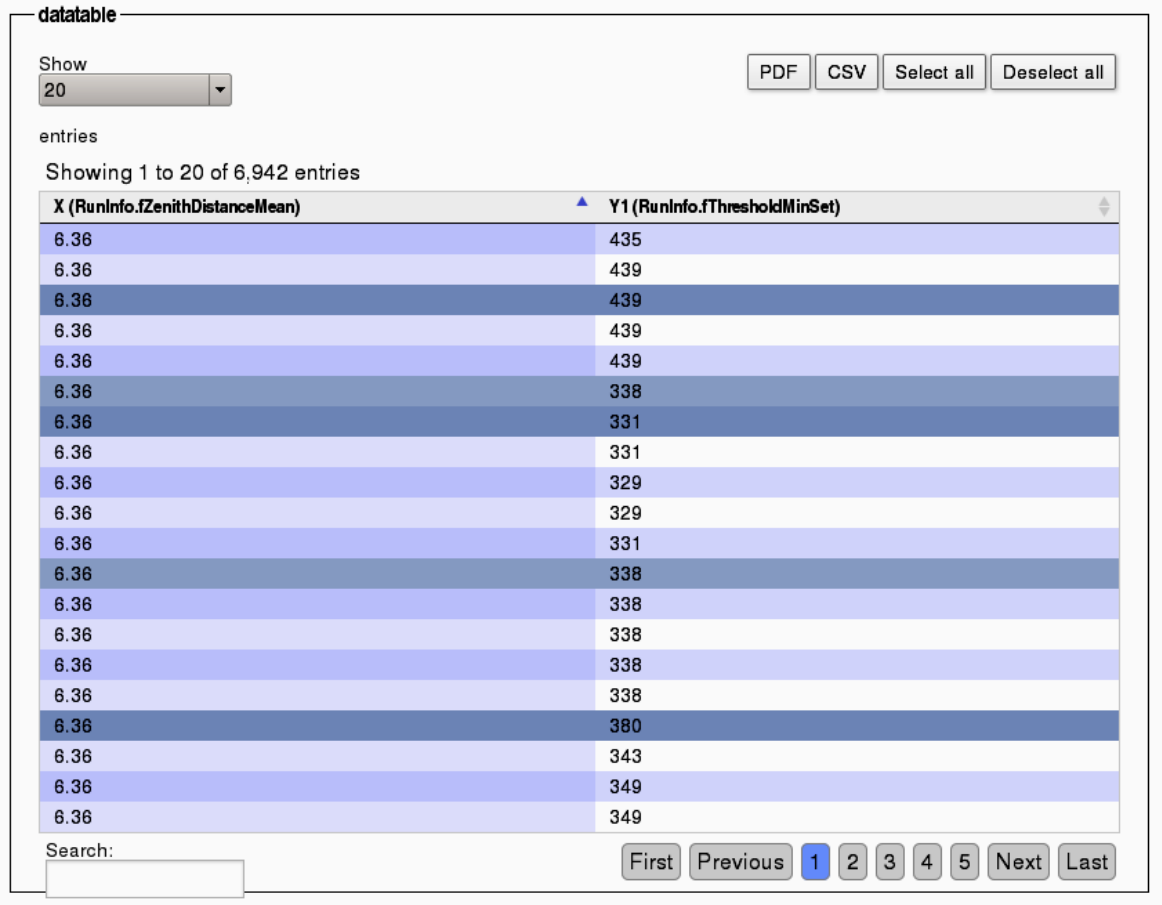


Figure 5.3: Screenshot of the website. In the datatable, selected rows are highlighted with a darker colour.

When at least one row is selected, the "plot" button can be pressed and the chosen data are plotted.

6 The Plotting Tools: Highcharts and jqPlot

As soon as the plot button is pressed, the function *plotting()* is executed. Analoguely to *ajax()* it deletes an already existing plot by removing the div it is displayed in. Then the data from the selected rows in the datatable has to be prepared for the plotting tools. This is done with the help of the function *createData()* which turns the row data into a JavaScript array.

```
function plotting() {
 $('#highcharts div').remove('#plot');
 $('#highcharts').prepend('<div id=\"plot\" style=\"width: 700px;
    height: 500px;\"></div>');
 var values = createData(table);
 //...

}
```

6.1: Function plotting()

Since the complete code of the *createData()* function would be too much to display it here, I describe what it does.

- First all selected rows of the datatable are stored in a JavaScript array.
- Then it checks, whether this array is empty (not a single row is selected). If this is the case, a boolean value is set to false and the function terminates by telling the user via alert window to select at least one row.
- The third task is to extract all values for each column and put them into an array. But before this is done, a check is performed to find out whether this column is an additional column (which means it should not be plotted). If this is not the case, the values are stored in an array.
- Fourth, every data cell is checked for content, because the query can also return NULL values, which can not be plotted, of course. These values are counted and the user gets an alert window with the number of filtered out NULL values.

- Next, the number of additional columns has to be determined to get the actual number of columns to plot.
- Finally, a multidimensional array with all valid values is returned.

Now the actual number of columns is calculated by subtracting the number of additional columns. If the multidimensional array has no entries (because one column consists only of NULL values), an alert window is shown and the function terminates. Since the individual values of the subarrays are not sorted, it is necessary to sort at least one of them - in this case the first subarray is sorted. This is not an easy thing to do, but the adapted code from Sean Kinsey [14] is a very good and short solution for this problem:

```
var a = ... // multidimensional array
a.sort((function(index) {
 return function(a, b) {
  if (a[index] === '') {
   return 1;
  } else {
   return (a[index] === b[index] ? 0 : (a[index] < b[index] ? -1 : 1));
  }
 };
})(0)); // index of the subarray which should be sorted
```

6.2: Sorting of multidimensional arrays

Now the first subarray is sorted and the data are ready to be plotted.
During the development of the website, I realized that some queries can return up to 80,000 data points. Before that problem turned up, I was planning to use *Highcharts* [4], which is a very good plotting tool especially when it comes to interacting with the displayed chart. But unfortunately, creating a plot with Highcharts out of more than 10,000 data points renders the website inacceptibly slow. Looking for other JavaScript plotting tools, I came across *jqPlot* [9]. It shares many features with Highcharts **and** can easily deal with up to 50,000 data points. To not lose the advantages of both tools, I implemented both letting the user choose. The div containing the plot has the id "highcharts" and contains per default a button to reset the jqPlot zoom and two selection boxes – one for the plotting tool and one for the plot type. The html code is the following:

```
<div class="large-8 columns">
 <fieldset style="margin-right: -65px;">
  <legend>plot</legend>
  <div id="highcharts">
   <div id="plot"> </div>
   <div style="padding-top:20px;">
    <input type="button" id="res" value="reset jqplot zoom" disabled><
       br />
```

```
    Please select a plotting tool:
    <select class="sel1" style="width: 100px;">
     <option value="jq">jqplot</option>
     <option value="high">highcharts (not more than 10,000 data points)
        </option>
    </select>
    and a plot option:
    <select class="sel2" style="width: 100px;">
     <option value="scatter">scatter</option>
     <option value="graph">graph</option>
     <option value="histo">histogram</option>
     <option value="bubble">bubble(highcharts)</option>
    </select>
   </div>
  </div>
 </fieldset>
</div>
```

6.3: Graph options

This is how it looks on the website:

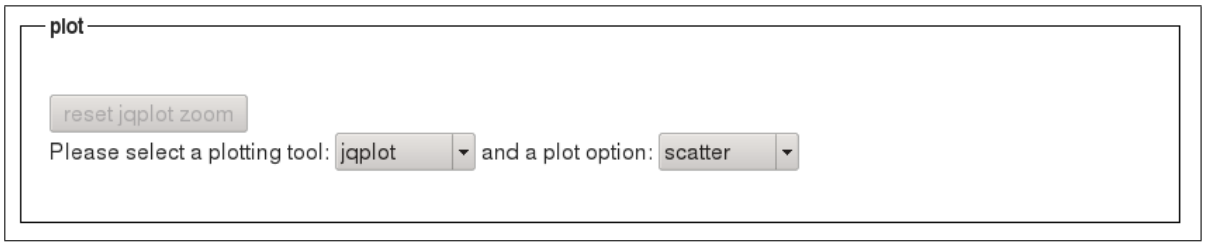


Figure 6.1: Screenshot of the website. The plot fieldset contains per default the (here disabled) reset zoom button for jqPlot, and two drop down menus – one for the plotting tool and one for the plot type.

Since Highcharts and jqPlot are two different plotting tools, the plot type options (e.g. line, scatter) need to be defined differently. For Highcharts, the type is specified with the variable defaultSeriesType and its value is a string. For jqPlot, the type definition is realized by so called "renderers" (e.g. LineRenderer, BarRenderer) – with an exception for the scatter type, which is a string. So the plotting function (see appendix, 4) needs to find out which type of plot is chosen and then sets these variables to the corresponding values. The type "bubble" is not defined for jqPlot and can only be used in Highcharts plots. When the types are defined, the plotting tool is determined and depending on the tool, a different option is called.

6.1 jqPlot

When jqPlot is chosen, the function *plotJQ()* is executed. Since this plotting tool does not offer a "reset jqplot zoom"-button by default I created one myself. It is enabled when jqPlot is chosen and disabled when Highcharts is chosen. Whenever a user clicks on that button, the jqPlot function *resetZoom()* is executed.

```
var jQPlot;
$('#highcharts #res').click(function() {
 jQPlot.resetZoom();
});
```

The jqPlot itself is defined by three variables, namely a string, the data points stored in an array and the different options which are all defined in one variable "options":

```
var jQPlot;
function plotJQ(a, num, ren, s) {
 // enable reset zoom button
 $('#highcharts #res').prop('disabled', false);
 // ...
 // options for jqPlot
 var options = ...
 // ...
 //
 jQPlot = $.jqplot('plot', [a], options);
}
```

6.4: Function plotJQ()

Unfortunately, jqPlot does not offer the opportunity to define the options once and then simply extend the data array to several subarrays (which means several y-axes in the plot). For every number of subarrays the options have to be redefined, which results in many lines of code. This is why I limited the maximum numbers of y axes to be plotted to five. In the jQuery code in the appendix, 5, I give an example of the options for a plot with one x- and one y-axis.
So if the user chooses the jqPlot option, the plot style will look similar to the one seen in Figure 6.2.

6.2 Highcharts

Analoguely to jqPlot, the Highcharts plot is a variable and certain options need to be defined. When the function *plotHigh()* is called, the first action is the disabling of the "reset jqplot zoom"-button. Then some general options for displaying numbers on the axes are specified. For the initialisation of the plot variable one simply needs to define the options for the x axis and one y axis once. Two Highcharts functions are used:

- *addSeries()* defines a new array of data points

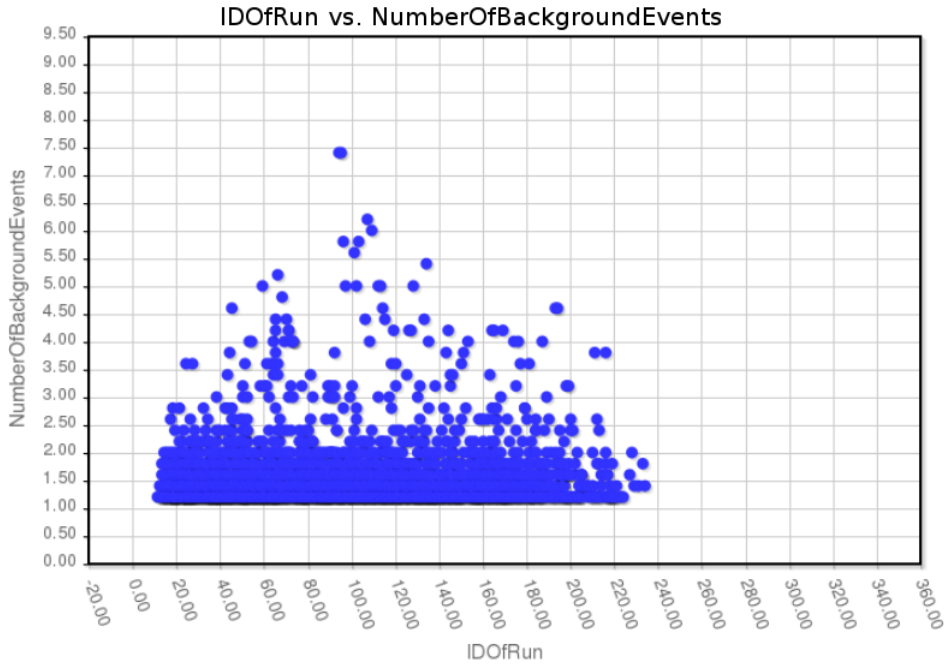


Figure 6.2: Screenshot of the website. A typical look of a jqPlot plot can be seen.

- *addAxis()* defines a new axis

So the number of axes is checked and based on this, the corresponding number of series and axes is added. The complete function *plotHigh()* can be found in the appendix, 6. If the user chooses the Highcharts option, a plot will have this style:

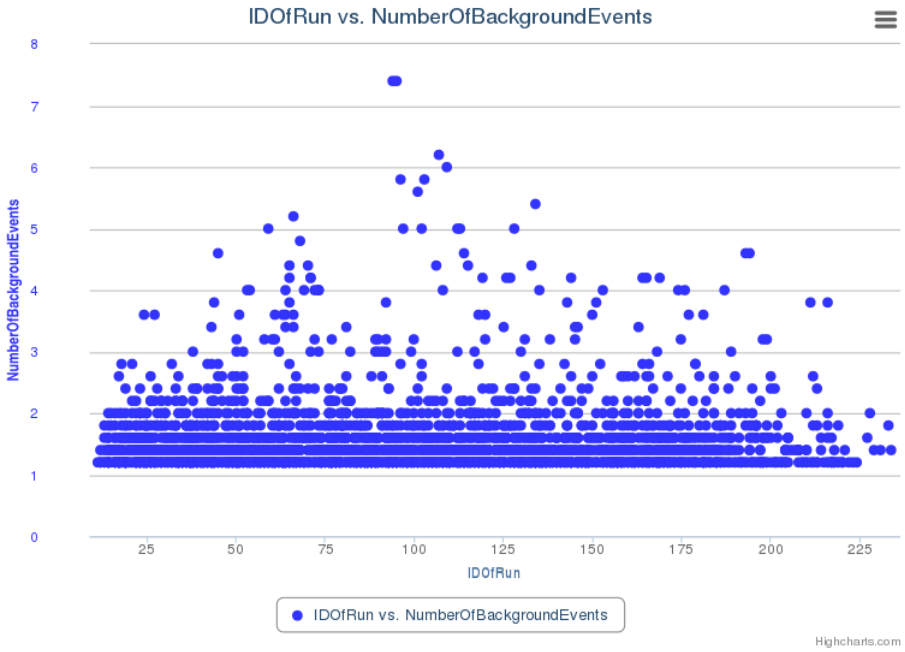


Figure 6.3: Screenshot of the website. A typical look of a Highcharts plot can be seen.

6.3 Changing Plot Options

As seen in Figure 6.1, the default plotting tool is jqPlot and the default plot type is scatter. If one of these options is changed, the plot has to be redrawn. To check whether an option is changed, jQuery offers the function *change()*:

```
// user changed plotting tool
$('#highcharts .sel1').change(function() {
 if (isSelected) {
  plotting();
 }
});
```

6.5: Change of plotting tool

To prevent the user from clicking on the "plot"-button without having sent a query, the boolean variable isSelected is checked before *plotting()* is executed. isSelected is set to true in the *createData()* function when it is made sure, that at least one row from the datatable has been selected.

If the user changes the plot type, 6.5 is executed analoguely with

```
$('#highcharts .sel2').change(function() {
// ...
}
```

Now the website provides all tools necessary for a data check and some examples can be given.

7 The Datacheck: Example Plots

Of course, the number of tables and columns from the FACT database is not small and many examples for data check plots are possible. When the user wants to do a data check, the tables *AnalysisResultsRunISDC* and *RunInfo* are most important. Since such a data check requires a lot of cross checking with the FACT logbook [16], I will only focus on these two tables and give one example on how to perform such a check. This example reveals the most typical disturbances that make such a data check necessary.

7.1 Threshold Vs. Rate After Quality Cuts

7.1.1 Threshold

Before the example is given, I will give a short explanation of the term "threshold" and "rate after quality cuts". In the FACT data acquisition system, always the last signal is buffered. The threshold is a limit of the sum on a signal in a cluster of 9 pixels of the camera. Whenever this limit is exceeded, the signal is read and recorded as an event. A big influence on this comes from atmospheric conditions like clouds or high humidity and from light sources like the moon. The threshold is chosen such that the trigger rate is not dominated by background. For example, when the level of nightsky background light goes up, the threshold increases.

7.1.2 Rate After Quality Cuts

The rate after quality cuts is the number of events after quality cuts divided by the effective ontime. The number of events after quality cuts comprises all events surviving the image cleaning (i.e. removing all pixels from the image which do not contain signal) and a first set of cuts which remove all events that cannot be reconstructed (e.g. events with less than 5 pixels or events leaking out of the camera).

7.1.3 Performing the Datacheck

The example will be a plot of the threshold vs. the rate[1]. The first can be found in the table *RunInfo*, the latter can be constructed from *AnalysisResultsRunISDC*. First, some aliases are defined for each axis because this will result in a nice axes labeling of the plot. The aliases and axes input fields (see Figure 7.1) and the cuts (see Figure 7.2) are defined with the help of the drag and drop function.

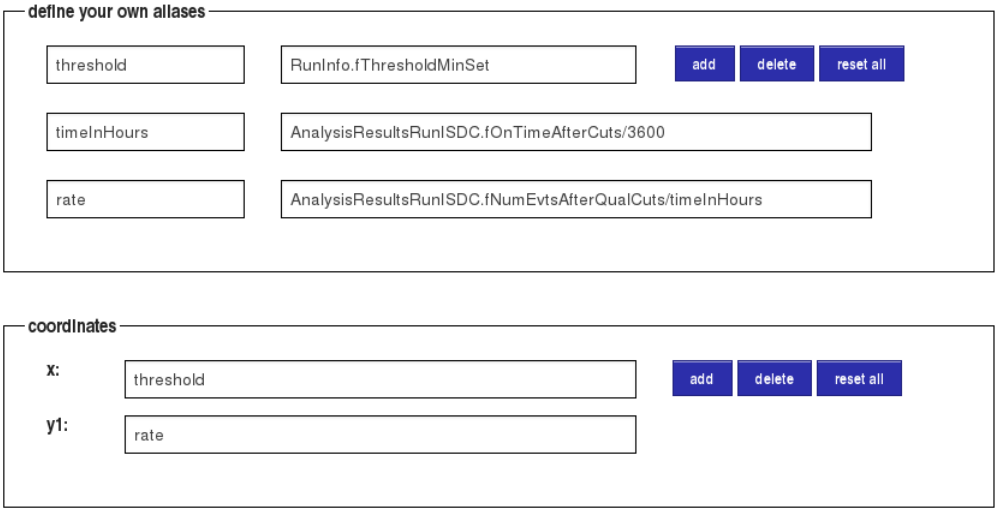


Figure 7.1: Screenshot from a part of the website. In the upper part, the aliases for the SQL statement are defined. In the lower part, the definition of the x and y axis is shown.

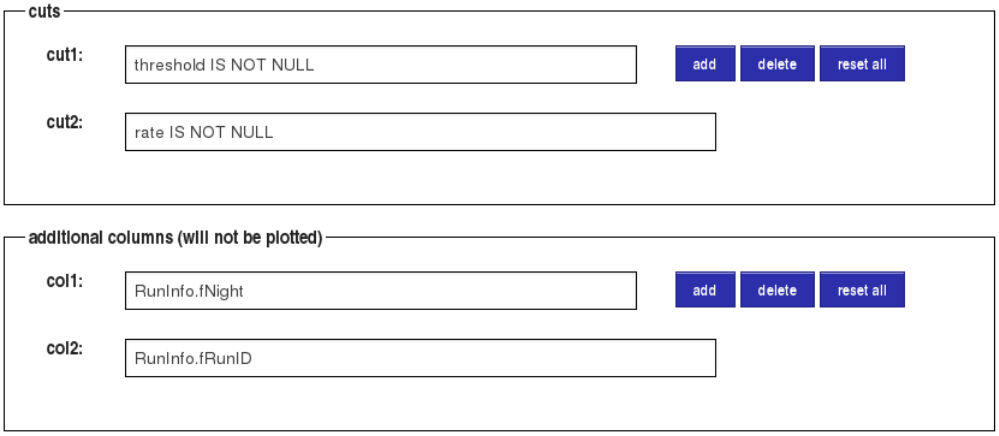


Figure 7.2: Screenshot from a part of the website. In the upper part, the cuts for the SQL statement are defined. In the lower part, the definition of additional columns can be seen.

[1] The rate is defined with: number of events after quality cuts divided by the ontime after cuts. Since the ontime entity is seconds, it has to be divided by 3600 to get hours.

To save calculation time, two cuts are defined which filter out the NULL values[2]. Since the user has to look at the logbook entries to remove some data points from the plot, the run id and the date (night) are displayed in the table as additional columns.

The created SQL statement is displayed in Figure 7.3.

sql-statement

```
SELECT (RunInfo.fThresholdMinSet) AS X, (AnalysisResultsRunISDC.fNumEvtsAfterQualCuts/
(AnalysisResultsRunISDC.fOnTimeAfterCuts/3600)) AS Y1, RunInfo.fNight, RunInfo.fRunID FROM
AnalysisResultsRunISDC LEFT JOIN RunInfo USING (fNight,fRunID) WHERE (RunInfo.fThresholdMinSet) IS
NOT NULL AND (AnalysisResultsRunISDC.fNumEvtsAfterQualCuts/
(AnalysisResultsRunISDC.fOnTimeAfterCuts/3600)) IS NOT NULL;
```

Figure 7.3: Screenshot from a part of the website. The generated SQL statement as it is shown on the website.

Selecting all rows from the created datatable yields this plot (zoomed in):

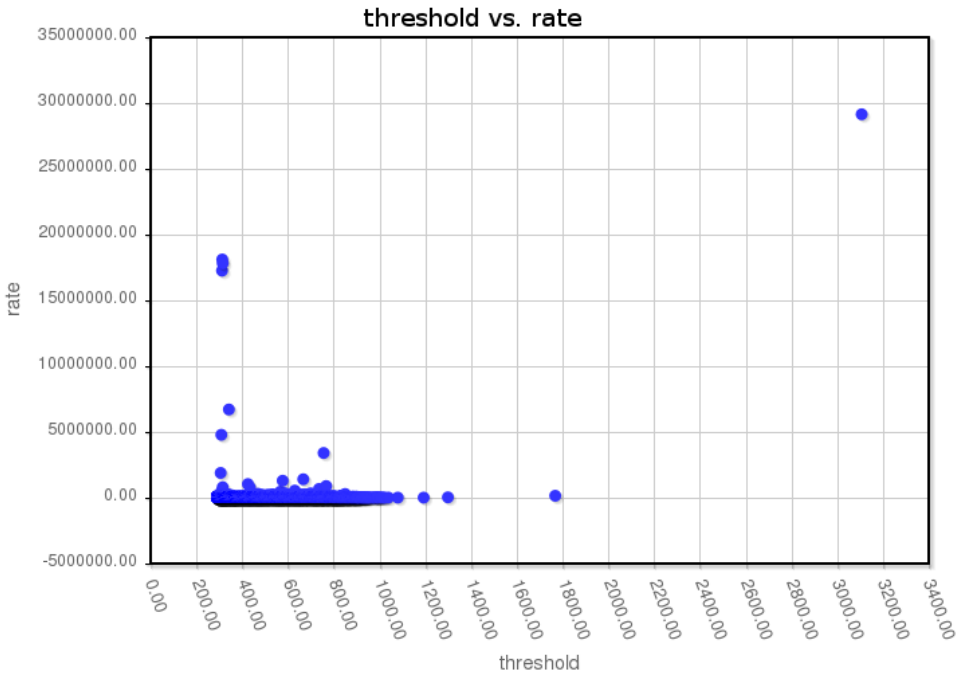


Figure 7.4: Screenshot from a part of the website. The first plot result from the example is shown as it is generated with jqPlot. The x axis shows the threshold, the y axis the rate after quality cuts.

One can see that most data points range between a threshold of 200 and 1,400 and (after zooming in) a rate between 0 and 200,000. But obviously, some data points differ

[2] NULL values are empty entries in the database. The website filters out NULL values anyway, but adding such a cut results in less data and thus less computing time.

greatly from the others. Sorting the datatable by the x axis values, the additional columns in the table show the corresponding nights and runs (see Figure 7.5).

Showing 1 to 20 of 20,548 entries

X (threshold)	Y1 (rate)	COL1(RunInfo.fNight)	COL2(RunInfo.fRunID)
3107	29147586.0471419320	20121026	26
1768	151018.9278218128	20121004	51
1299	32250.3401294877	20131018	78
1193	8969.0726352481	20131018	89
1080	10345.4595891680	20131014	122
1039	10886.6299056364	20130323	61
1023	8.1360006356	20130323	21
1017	12490.7655067616	20130621	23
1008	16447.3680541419	20131014	12
1005	12805.1739260660	20130621	31
997	20208.4481212668	20131014	13
991	20931.7679827256	20131014	14

Figure 7.5: Screenshot from a part of the website. The first part of the generated datatable is seen, sorted by the x axis values. The additional columns RunInfo.fNight and RunInfo.fRunID appear here, but are not plotted. This additional information eases the data check.

The night for the first point in this table is October 26, 2012 and the run has the id 26. Checking the entries from this night in the FACT Logbook [15] reveals a typical source of disturbance for the data taking: FACT is not the only telescope on Roque de los Muchachos (in total, 15 are placed there) and other telescopes like the William Herschel Telescope often use *lasers* (see Figure 7.6) during moonlight as a guide to point

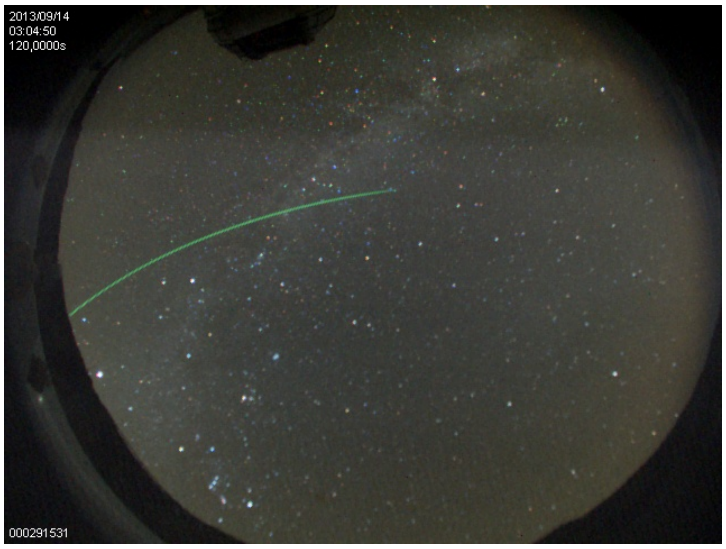


Figure 7.6: Picture from the allsky camera of the Gran Telescopio de Canarias located on Roque de los Muchachos, La Palma, taken on September 14, 2013. The green line in the left half of the picture shows a green laser ray, coming from Wilhelm Herschel Telescope nearby. When this laser is in the field of view of the FACT camera, the data are distorted.

at the sources they want to observe. This laser can be in the field of view of FACT and thus distort the data. The next data point in the table was taken on October 4, 2012 with run id 51. In this run, the shifter had to stop the data taking script because of too high currents. The reason for the high currents was the moon in combination with clouds (see Figure 7.7), which scatter the moonlight and the sky appears even brighter.

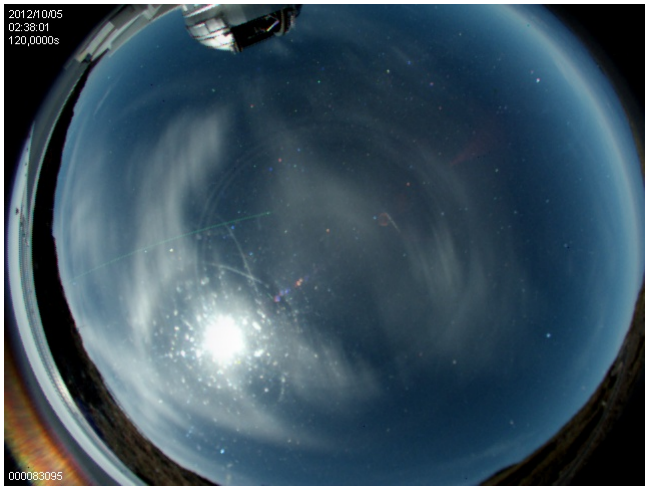


Figure 7.7: Picture from the allsky camera of the Gran Telescopio de Canarias , taken on October 5, 2012. Wisps of clouds are seen as well as the moon. The clouds scatter the moonlight and the sky appears brighter.

The following three data points (taken on October 18 and 14, 2013) share this problem and can be deselected, too. After the deselections, the data is replotted:

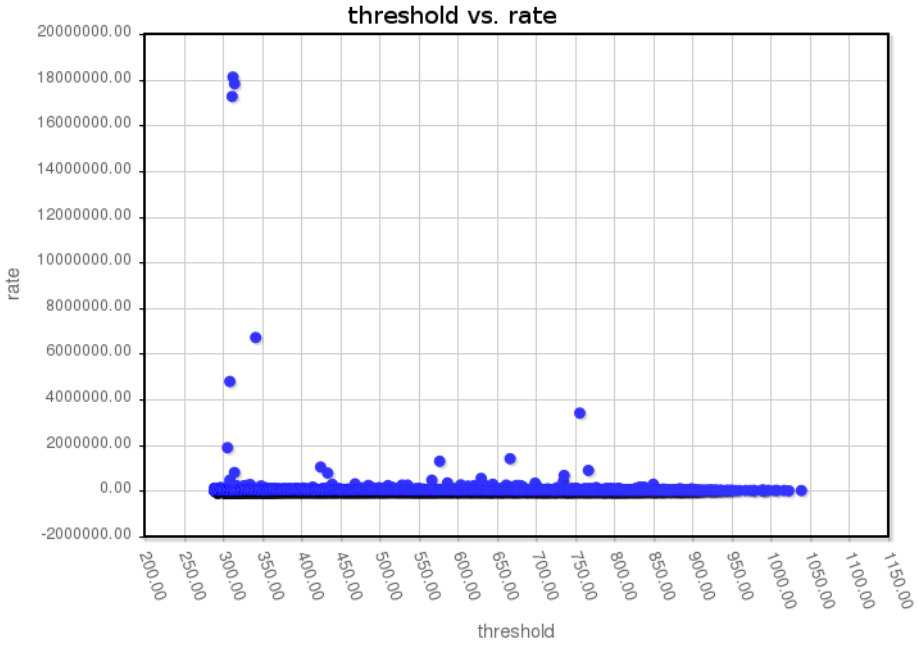


Figure 7.8: Screenshot of the website. Plot of the rate after quality cuts vs. threshold. After the first data check, some points were removed from the data set.

Showing 1 to 20 of 20,548 entries

X (threshold)	Y1 (rate)	COL1(RunInfo.fNight)	COL2(RunInfo.fRunID)
3107	29147586.0471419320	20121026	26
313	18111219.2313462700	20130913	140
315	17817739.8037007150	20130913	141
312	17260986.4371557570	20130913	151
342	6709145.6429033180	20130513	38
309	4785365.8953995170	20130119	138
756	3402516.0566371363	20120925	23
306	1886666.6611133780	20130210	123
667	1405479.4153323392	20130320	14
577	1292999.0616664276	20121026	48
425	1039636.3786587047	20130501	21

Figure 7.9: Screenshot from a part of the website. The first part of the generated datatable is seen, sorted by the y axis values.

Now the table is sorted by the y axis (see Figure 7.9). Again it is easy to see, that the first data points seem to differ greatly from the rest. The first data point is the same as in the table sorted by x values (see Figure 7.5), so it can be deselected. What catches the eye, is the fact that the next three data points in the table were all taken the same night, namely September 13, 2013 and their run numbers are close together. Looking up the corresponding entries in the logbook, we can deselect these three points for the same reason the first one is excluded: laser in the field of view. Checking the entries for the next data point, the shifter reports a permanent light source hitting the camera (see Figure 7.10). Of course, this light source distorts the data and the data point must be excluded from the plot.

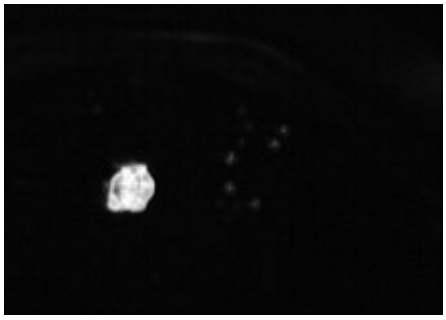


Figure 7.10: Picture from the CCD camera located in the FACT mirror dish and facing the FACT camera, taken on September 13, 2013. There is a light source in camera which affects the data.

The sixth data point in the table was taken on January 19, 2013. Due to a incorrect DRS chip calibration and a software problem, this run can be deselected, too. The seventh point, taken on September 25, 2012, can be deselected due to too high currents. So after deselecting the first seven points from the table in Figure 7.9, a new plot can be generated (zoomed in, see Figure 7.11).

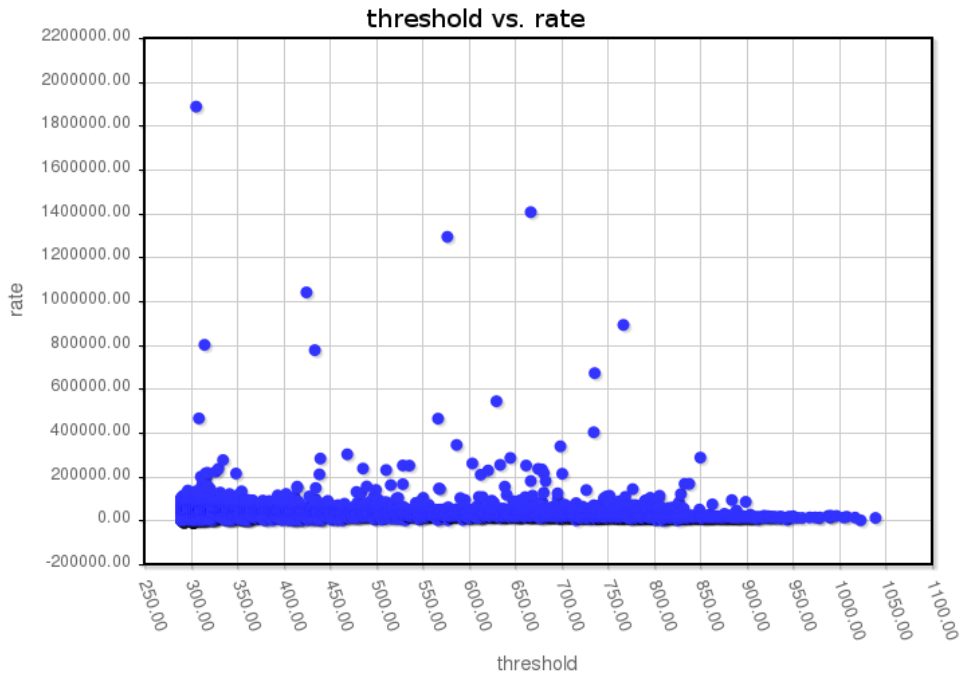


Figure 7.11: Screenshot of the website. Plot of the rate after quality cuts vs. threshold. The second step of the data-check is applied and more points were removed from the data set.

Showing 1 to 20 of 20,548 entries

X (threshold)	Y1 (rate)	COL1(RunInfo.fNight)	COL2(RunInfo.fRunID)
3107	29147586.0471419320	20121026	26
313	18111219.2313462700	20130913	140
315	17817739.8037007150	20130913	141
312	17260986.4371557570	20130913	151
342	6709145.6429033180	20130513	38
309	4785365.8953995170	20130119	138
756	3402516.0566371363	20120925	23
306	1886666.6611133780	20130210	123
667	1405479.4153323392	20130320	14
577	1292999.0616664276	20121026	48
425	1039636.3786587047	20130501	21
767	891254.6934321420	20120925	125
315	800303.8927037326	20130913	139
434	776099.9553481406	20121202	22
736	671278.1738284708	20121221	27

Figure 7.12: Screenshot from a part of the website. The first part of the generated datatable is seen, sorted by the y axis values. The first data points are deselected.

Still, there are points with very high rates above 400,000 (see Figure 7.12). The next data point in the table was taken on February 10, 2013 and has the run id 123. In the logbook, the shifter mentions that this run is affected by "lidar". These are laser shots from the nearby MAGIC telescope to do atmospheric measurements. These shots are in the field of view of the FACT camera (see Figure 7.13) and produce additional light

which results in a higher event rate. So this lidar affected run can also be taken from the data set by deselection.

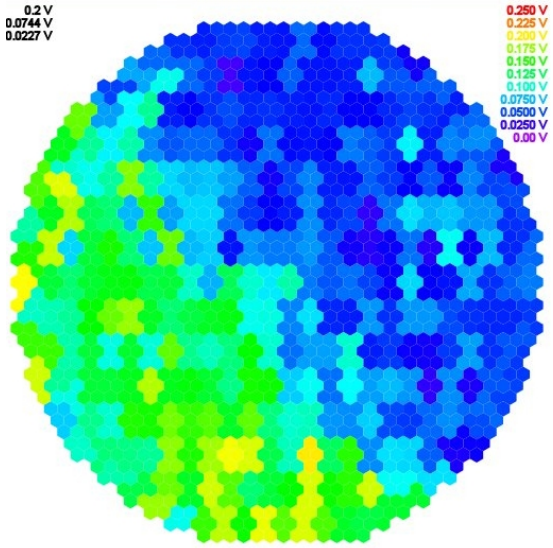


Figure 7.13: Screenshot from SmartFact[3]. A typical pattern for lidar shots is seen in the left half of the camera (green/yellow).

Another reason for excluding data points from the set can be technical or software problems, e.g. the script was stopped during a run or the data taking procedure was interrupted by FAD losses. Especially in the early phase of the data taking with FACT, the last runs of the night were taken when the sun was about to rise. Since this bright light causes a higher rate, these points will be excluded, too.

Continuing this procedure a little bit further for data points that do not seem to fit into the plot, I found out that all points with x values greater than 166,000 can be removed from the data set. At the point I stopped the data check, the plot revealed a form of relation between the threshold and the rate after quality cuts (see Figure 7.14). The data seem to form a descending curve: when the rate is high, the threshold is low and vice versa. But still some datapoints seem to form a bulk between a rate of approximately 40,000 and 90,000 and a threshold of approximately 600 to 880 (see Figure 7.14, red ellipse). Looking at the nights of these data points, one can see that all of them were taken before March 1, 2013 whereas all data points within the green ellipse were taken after this date. The reason for these two distributions in the plot is a change in the ratecontrol algorithm which took place at this point of time. Before March 1, 2013, the rate was adjusted to a fixed value, afterwards it is calculated with the help of the distribution threshold vs. currents.

[3]Online display to monitor the FACT data taking.

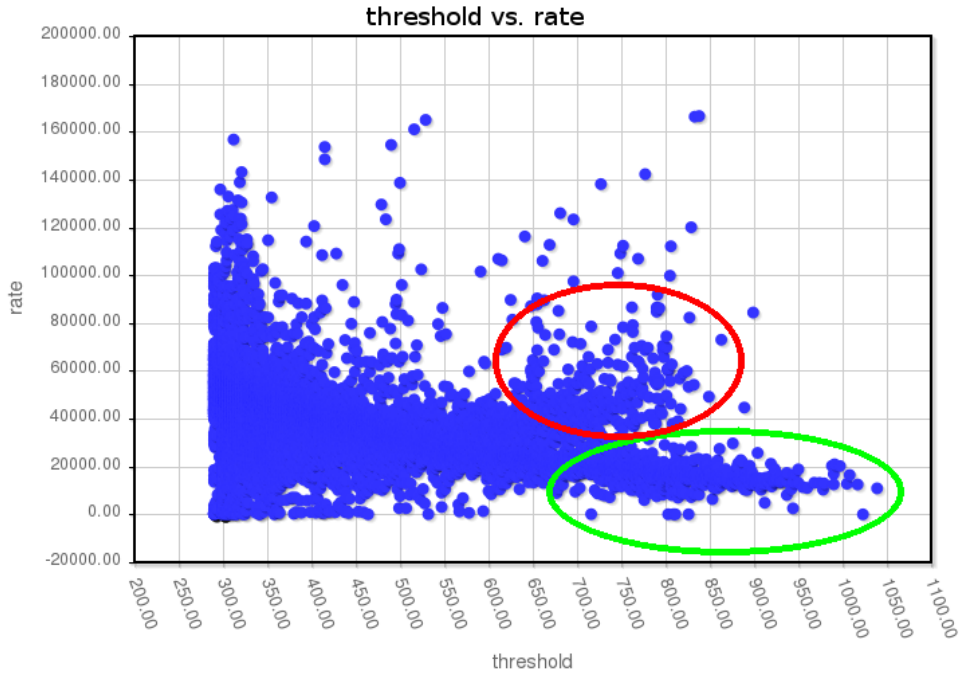


Figure 7.14: Screenshot from a part of the website. Plot of the rate after quality cuts vs. threshold. The third step of the data-check is applied. All points with a rate smaller than 166,000 and a threshold smaller 1,050 are displayed. Marked with green andred are two distributions which can be attributed to before and after a change in the algorithm of the ratecontrol.

7.2 Nightly Excess Rate Curves

Since the database contains all information about the data taking, the website can also be used to plot so called "nightly excess rate curves" of the observed sources.

7.2.1 Excess Rate

As mentioned in Chapter 1.2.1, the telescope uses a wobble mode to simultaneously determine the flux of background events. In one wobble mode, one "on region" and five "off regions" (all these six regions are placed symmetrically around the center of the camera with a distance of 0.6 degrees to each other and to the camera center) are used. Like this, on and off regions are measured simultaneously. For these regions and for each event the reconstruction of the source position is put into a histogram. The excess is then calculated by subtracting the background from the signal in the corresponding source region.

7.2.2 Nightly Excess Rate Curve of Mrk501

Whenever a flare (i.e. an increase in the flux) of an AGN happens, more particles from the source hit the Earth. This means that more excess events are detected and the excess rate increases. One of the most often observed AGN is one nearest to us: the blazar Markarian (Mrk) 501. As described in Chapter 1.1.1, blazars show two peaks in their SEDs: one in the gamma- and one in the X-ray band. Whenever a flare happens, some theoretical models excpect that the flux in these two peaks increases simultaneously. Since FACT is only able to detect these flares in the gamma-ray band, it is not possible to have simultaneous X-ray data whenever a flare happens. In 2012, I came up with the idea of writing a proposal to an X-ray telescope to monitor Mrk501 simultaneously whenever FACT detects an obvious increase of the flux in the gamma-ray band. The X-ray telescope I chose was the X-ray Multi-Mirror Mission (XMM-Newton) telescope [6], a telescope operated by the ESA on board of a satellite. In order to be able to conduct such a simultaneous observation, both FACT and XMM-Newton must be able to see the source. Doing some calculation, two time windows were chosen: one from July 12, to August 31, 2013 and one from March 1, 2014 to April 1, 2014. The chance for this proposal to be allocated observing time with XMM-Newton was 1:10. But two months after submitting the proposal, I got the news that it was actually accepted. So we observed Mrk501 with FACT in July and August whenever possible but unfortunately, there was no flare. With the help of the website, one can plot the nightly excess rate for this time range. To get the number of all excess events in one night, the sum of the number of all excess events in one run divided by the ontime is divided by the number

of runs. In order to get a nice axes labeling, some aliases are defined (see Figure 7.15).

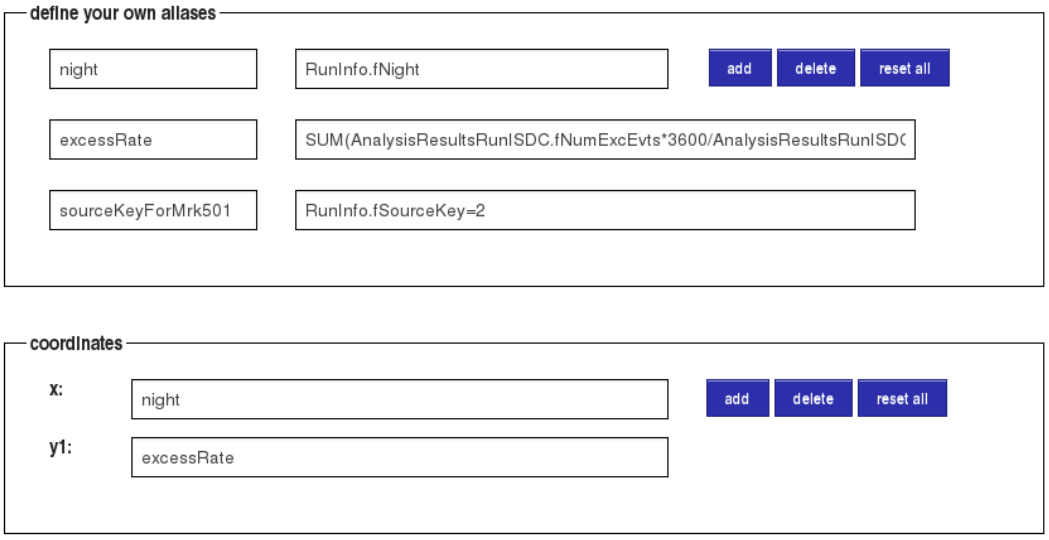


Figure 7.15: Screenshot of the website. The aliases and coordinates for the nightly excess rate plot of Mrk501 can be seen. With the help of the SQL command SUM(), the sum can easily be calculated.

In order to get the time range from July 1, 2013 to August 31, 2013, cuts have to be defined. Figure 7.16 shows the cuts for this time range, the cuts for the second time range in August were defined accordingly.

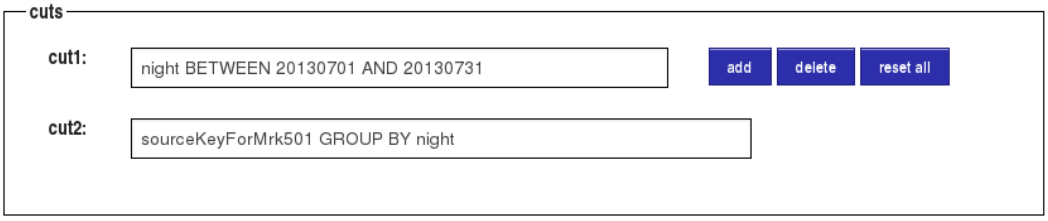


Figure 7.16: Screenshot of the website. The cuts for the nightly excess rate plot of Mrk501 for the time range July 1, 2013 to July 31, 2013 are displayed. The SQL command GROUP BY is added to display the individual nights.

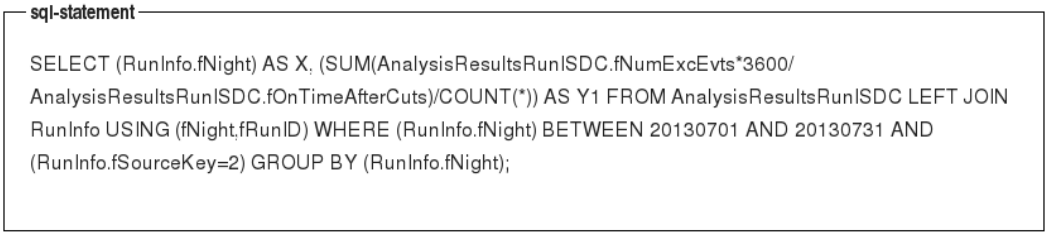


Figure 7.17: Screenshot of the website. The SQL statement for the nightly excess rate plot of Mrk501 from July 1, 2013 to July 31, 2013 is displayed.

In the plots (see Figures 7.18 and 7.19) from the generated SQL statements (see Figure 7.17, statement for second time range was done accordingly) one can see that the highest rates were recorded on July 18, 2013 and August 25, 2013.

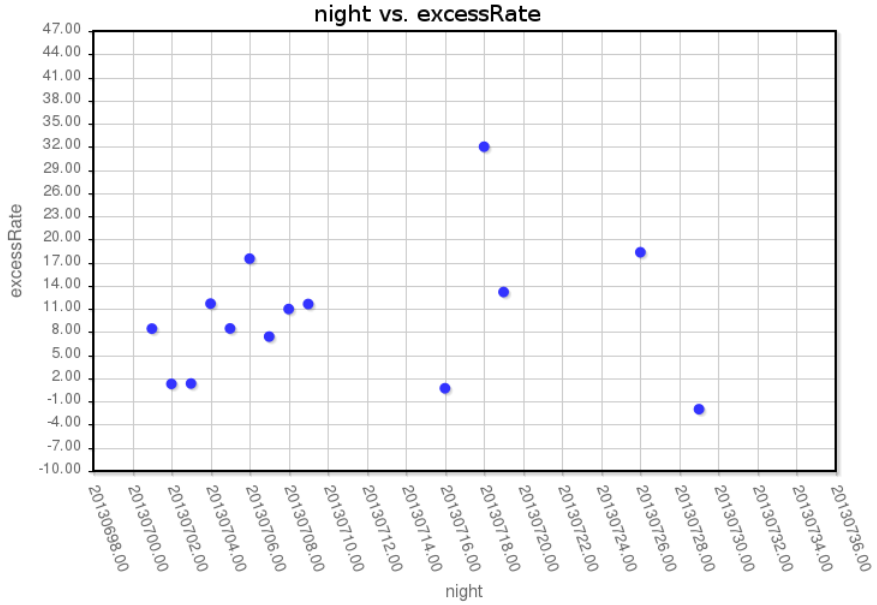


Figure 7.18: Screenshot of the website. The nightly excess rate plot of Mrk501 for the time range July 1, 2013 to July 31, 2013 is displayed.[4]

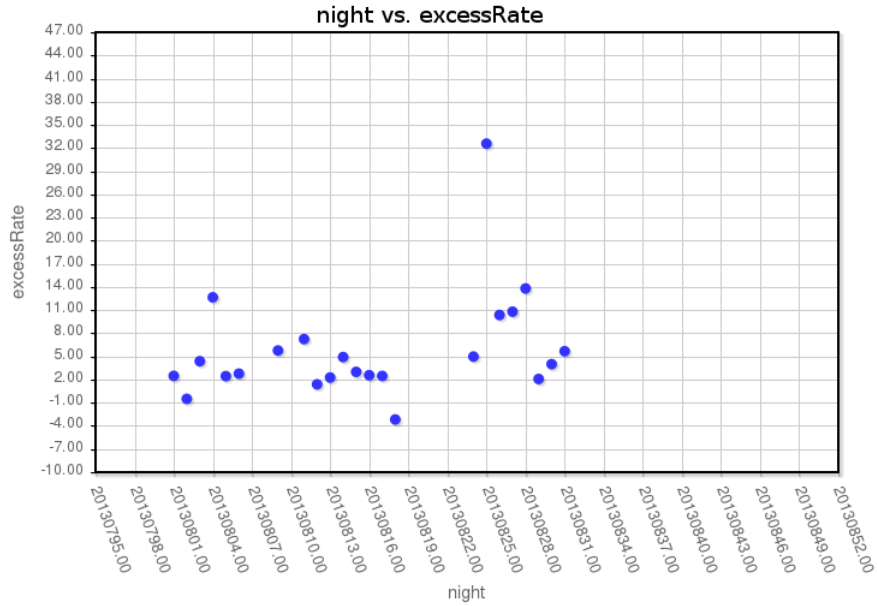


Figure 7.19: Screenshot of the website. The nightly excess rate plot of Mrk501 for the time range August 1, 2013 to August 31, 2013 is displayed.

[4]The date is stored in the form yyyymmdd (e.g. 20130107) in the database which why the plotting tool regards this as a number and simply continues counting such that 20130732 follows after 20130731.

To have a good comparison, I plotted the data from June 1, 2012 to June 30, 2012 because Mrk 501 showed a flare on June 8, 2012. The plot (see Figure 7.20) shows a high excess rate of 106.09 on June 8. So compared to the highest peaks from the 2013 plots (excess rate of 32.01 resp. 32.57), this value is much higher and a real flare took place.

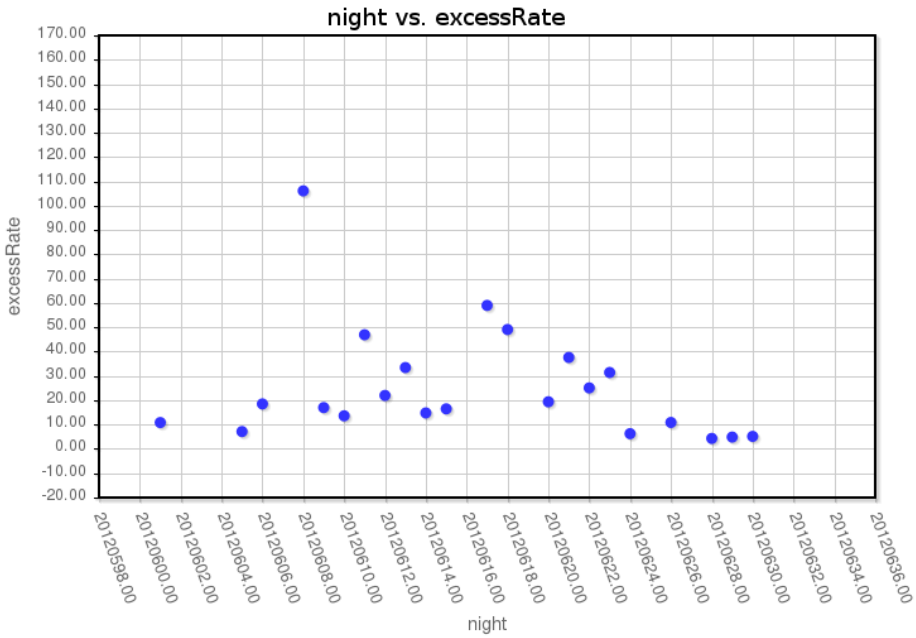


Figure 7.20: Screenshot of the website. The nightly excess rate plot of Mrk501 for the time range June 1, 2012 to June 30, 2012 is displayed.

So right now I am hoping, that FACT can detect a flare in the time from March 1, 2014 to April 1, 2014 so that we can trigger a simultaneous XMM-Newton observation.

8 Summary and Outlook

The goal of this thesis is to develop a web interface for the FACT collaboration in order to provide a userfriendly tool to perform a data check. The website must provide all necessary tools to send an SQL statement to the FACT database and create a plot from its result. To prepare, I had to get acquainted with all necessary tools for web programming. A big help were two books from the "Head First"-book series, which provide a very good introduction to HTML and CSS [19] and the JavaScript library *jQuery* [2]. I also had to get in touch with PHP, for which a very good documentation can be found online [17]. With the help of *The ZURB Foundation Framework* [8], the generation of the layout was not too difficult and the usage of *jQuery* instead of pure JavaScript proved a save of many lines of code. Taking *jQuery DataTables* as the plugin to display the result of a query was a very good choice, since it provides all necessary features (e.g. row (de-)selection, search function) contributing to a feasible data check. However, during the development of the website, I encountered problems which originated mostly from JavaScript variables, since they are not strictly bound to type definitions like in other programming languages (e.g. Java). A bigger problem was the finding of a JavaScript plotting tool, which is capable of dealing with a large amount of data points in a reasonable time. From the collaboration, the tool *Highcharts* [4] was recommended, but unfortunately, it is not usable for more than 10,000 data points. Investigating different alternatives, I came across *jqPlot*, which provides the same basic functions as *Highcharts* (except saving the plot in a file), but can easily deal with up to 50,000 data points. But since *Highcharts* has a more professional look than *jqPlot*, I decided to let the user choose and implemented them both in the website. In the end, the tests of the website and the plotting tools revealed some last little errors in the code, which were fixed and I was able to perform data checks with the website and the help of the FACT logbook.

At this point, all user requirements from chapter 1.3 are fulfilled:

- The website provides the possibility to easily access the database, select data and plot them.
- Various input fields and buttons help in generating and submitting the SQL statement, as well as plotting the data.

- Long input field terms only have to be defined once as an alias in the corresponding input field, then they can be reused in other fields. Additionally, these aliases allow the user to set custom axes descriptions for the plot (see Figures 7.1, 7.4).
- The result of every generated SQL statement is displayed in a jQuery DataTable, which can be searched for certain values, allows row selection and deselection and the export of the data in a file (.csv and .pdf).
- The selected rows can be plotted with two different plotting tools (Highcharts and jqPlot). Both plotting tools allow different types of plots (e.g. graph, scatter), several y axes as well as zooming into the plot. Compared to jqPlot, Highcharts offers the possibility of saving the plot in a file (.png, .jpeg, .pdf and .svg). The main advantage of jqPlot is its capability of being able to deal with up to 50,000 data points within a few seconds.

With all these components working, one can think of additional features to be implemented in the website in the future. Some can already be specified:

1. So far, the user can only save the datatable and the plot in a file but there is no way to save the query or the input fields. To solve this, a "save"-button will be added, which saves the most important information in a "query database" with two different tables. The first table should contain at least the following entries:
   - user name
   - query id
   - query title
   - input fields (axes, cuts, additional columns)

   The second table will be in charge of all defined aliases:
   - query id
   - alias name
   - alias content
2. The query database gives the opportunity to not only save but also to load information from it into the website. This will be accomplished with a "load"-button, which checks the user name of the currently logged in user and provides a list of his/her saved query titles. After choosing a title, the aliases and input field values are directly inserted into their corresponding places and the query can be executed again.

3. Of course, the user must have the possibility to delete saved queries from the query database. This is done with the help of a “delete”-button. Again the user gets a list of his/her saved queries and can choose the one to delete.

4. Another very nice feature to be implemented is the possibility for users to load queries from *other* users. To achieve this, a dropdown list menu with all user names will be put next to the buttons. The user can choose another name and select a query from his/her list. Since these queries should not be changed or deleted, the “save”- and “delete”-buttons must be disabled. If queries from other users are to be changed or copied, the user must switch back to his own username, otherwise no changes and copies are possible.

5. If the data can be saved with a certain structure, it can be analysed with the FACT analysis software package MARS [3]. The resulting light curves can be used to study the variability and draw conclusions on its origin. So another improvement for the user would be a “file”-button to save the data in a file usable for MARS.

6. So far, the defined cuts are applied to all axis. An improvement would be the possibility to define cuts for the individual axes. This allows for example the study of different distributions in plots similar to the example plot.

Right now, the website provides a performant tool for data checks for the FACT analysis. All requirements are fulfilled and implementing all the listed additional features will optimize the website.

Bibliography

[1] Olvier Arscott. https://twitter.com/OliverArscott/status/387629149189201920, 2013.

[2] Ryan Benedetti. *jQuery von Kopf bis Fuß*. O'Reilly, 1st edition, 2012.

[3] T. Bretz and D. Dorner. MARS - CheObs ed. – A flexible Software Framework for future Cherenkov Telescopes. In C. Leroy, P.-G. Rancoita, M. Barone, A. Gaddi, L. Price, & R. Ruchti, editor, *Astroparticle, Particle and Space Physics, Detectors and Medical Physics Applications*, pages 681–687, April 2010.

[4] Highcharts Documentation. http://www.highcharts.com/docs, 2013.

[5] Donato et al. Hard x-ray properties of blazars. *Astronomy & Astrophysics*, 375:739–751, 2001.

[6] X-ray Multi-Mirror Mission European Space Agency. http://xmm.esac.esa.int/, 2013.

[7] Ariel Flesler. http://flesler.blogspot.de/2007/10/jqueryscrollto.html, 2013.

[8] The Foundation Framework. http://foundation.zurb.com, 2013.

[9] jqPlot. http://www.jqplot.com/index.php, 2013.

[10] jQuery API Documentation. http:/api.jquery.com/, 2013.

[11] jQuery API Documentation. http://api.jqueryui.com/, 2013.

[12] jQuery DataTable. http://datatables.net/, 2013.

[13] Matthias Kadler. Vorlesungsskript Extragalaktische Jets, 2012.

[14] Sean Kinsey. http://stackoverflow.com/questions/2824145/sorting-a-multidimensional-array-in-javascript, 2010.

[15] FACT Logbook. https://www.fact-project.org/logbook/, 2013.

[16] FACT Collaboration Logbook. https://www.fact-project.org/logbook/.

[17] PHP Manual. http://www.php.net/manual/de/, 2013.

[18] Paolo Padovani. An introduction to Active Galactic Nuclei. 1., 2013.

[19] Elisabeth Robson. *HTML und CSS von Kopf bis Fuß*. O'Reilly, 2nd edition, 2012.

[20] Gideon Sireling. http://flesler.blogspot.de/2007/10/jqueryscrollto.html, 2013.

[21] ToggleAccordion. http://jsbin.com/eqape/1/edit, 2013.

List of Figures

List of Code Examples

Appendix

```
<div class="row">
 <form>
  <fieldset>
   <legend>Personal Data</legend>
    <div class="row">
     <div class="large-12 columns">
      <label>Name</label>
      <input type="text">
     </div>
    </div>
    <div class="row">
     <div class="large-4 columns">
      <label>City</label>
      <input type="text">
     </div>
     <div class="large-4 columns">
      <label>Phone</label>
      <input type="text">
     </div>
     <div class="large-4 columns">
      <div class="row collapse">
       <label>Email</label>
        <div class="small-9 columns">
         <input type="text">
        </div>
       <div class="small-3 columns">
        <span class="postfix">.de</span>
       </div>
     </div>
    </div>
   </div>
  </fieldset>
 </form>
</div>
```

1: HTML code for fieldset

```
function arrayFlatten(array $array) {
 $flatten = array();
 array_walk_recursive($array, function($value) use(&$flatten) {
  $flatten[] = $value;
 });
 return $flatten;
}
// since $pri is an array in an array: "flatten" array
$flattened = arrayFlatten($pri);
```

2: Function arrayFlatten()

```
// name of all columns in the table
$contents = '<label class="lab">Join on: </label><br/>';
while ($row = $statement_get_table_columns->fetch()) {
 // if the current row appeares in the primary key array, it's a
    primary key
 // give it a special background color
 if (containsString($flattened, $row['COLUMN_NAME'])) {
  $contents.=sprintf('<li class="masterTooltip" title="drag and drop me
     !"
  style="background-color: red;" data-rowname="%1$s">%1$s<input type="
     checkbox"
  name="%s" class="check"></li>', $row['COLUMN_NAME'], $row['
     COLUMN_NAME'])."\n";
 } else {
  $contents.=sprintf('<li class="masterTooltip" title="drag and drop me
     !"
  data-rowname="%1$s">%1$s<input type="checkbox" name="%s" class="check
     "></li>',
  $row['COLUMN_NAME'], $row['COLUMN_NAME'])."\n";
 }
}
// print rows
printf("<ul class=\"selectable\" data-tablename=\"%1s\">%s</ul>",
   $tblname,
$contents);
```

3: Creation of the database columns

```
function plotting() {
 $('#highcharts div').remove('#plot');
 $('#highcharts').prepend('<div id=\"plot\" style=\"width: 700px;
    height: 500px;\"></div>');
 var values = createData(table);
 // find out which type of graph is selected
 var defaultSeriesType;
 // renderer for jqPlot
 var renderer;
 // scatter option for jqPlot
 var scatter;
 // get current choice of plot type
 var type = $('#highcharts .sel2').val();
 // graph
 if (type === "graph") {
  defaultSeriesType = 'line';
  renderer = $.jqplot.LineRenderer;
 // histogram
 } else if (type === "histo") {
  defaultSeriesType = 'column';
  renderer = $.jqplot.BarRenderer;
 // bubble
 } else if (type === "bubble") {
  defaultSeriesType = 'bubble';
 // scatter
 } else if (type === "scatter") {
  defaultSeriesType = 'scatter';
  scatter = "scatter";
 }
 // find out which plotting tool is selected
 var val = $('#highcharts .sel1').val();
 if (val === "high") {
  plotHigh(a, numberOfColumns, defaultSeriesType);
 } else if (val === "jq") {
  plotJQ(a, numberOfColumns, renderer, scatter);
 }
}
```

4: Extended function plotting()

```
var jQPlot;
function plotJQ(a, num, ren, s) {
 // enable reset zoom button
 $('#highcharts #res').prop('disabled', false);
 // options for jqPlot
 var options = {
  // title of the plot
  title: 'Your selected data',
  // colors for the first five axes
  seriesColors: ["#3333FF", "#006666", "#E80000", "#990099", "#839557"
     ],
  // plot type options
  seriesDefaults: {
   renderer: ren,
   showShadow: false,
   showLine: show
  },
  // white background and black frame
  grid: {
   background: '#FFFFFF',
   borderColor: '#000000'
  },
  // offset for the axes labels
  legend: {
   xoffset: 10, yoffset: 10
  },
  // different axes options
  axesDefaults: {
   labelRenderer: $.jqplot.CanvasAxisLabelRenderer,
   tickRenderer: $.jqplot.CanvasAxisTickRenderer,
   numberTicks: 20,
   tickOptions: {
    fontSize: '10pt'
   }
  },
  // show tooltip with current data point coordinates
  highlighter: {
   show: true,
   sizeAdjust: 7.5
  },
  // enable zooming
  cursor: {
   show: true,
   zoom: true
  },
  // x axis definitions
  axes: {
   xaxis: {
    autoscale: true,
    label: columnNames[0],
    tickOptions: {
     formatString: '%.2f',
     angle: 70
    }
```

```
    },
    // y axis definitions
    yaxis: {
     autoscale: true,
     label: columnNames[1],
     tickOptions: {
      formatString: '%.2f'
     }
    }
   }
  };
  // plotting by calling $.jqplot()
  jQPlot = $.jqplot('plot', [a], options);
  // ...
}
```

5: Extended function plotJQ()

```
var highchartPlot;
function plotHigh(a, num, def) {
 // disable jqPlot zoom button
%  $('#highcharts #res').prop('disabled', true);
 Highcharts.setOptions({
  lang: {
   // no space between thousands (i.e. 10000 instead of 10 000)
   thousandsSep: '',
   // point instead of comma (i.e. 10.45 instead of 10,45)
   decimalPoint: '.'
  }
 });
 // initialize variable
 highchartPlot = new Highcharts.Chart({
  chart: {
   // place in plot div
   renderTo: 'plot',
   // type: line or column
   defaultSeriesType: def,
   // zoom in x and y
   zoomType: 'xy'
  },
  // title for graph
  title: {
   text: 'Your selected data'
  },
  // tooltip options
   tooltip: {
    shared: true,
    // create tooltip string
    formatter: function() {
     // x axis name and value
     var s = columnNames[0] + ": " + this.x;
     // y axes names and values
     $.each(this.points, function(i, point) {
      s += '<br/>' + columnNames[i + 1] + ": " + point.y;
      i++;
```

```
      });
      return s;
     }
    },
    // x axis style
    xAxis: {
     title: {
      text: columnNames[0]
     },
     labels: {
      formatter: function() {
       return this.value;
      }
     }
    },
    // y axis style
    yAxis: {
     offset: 30, title: {
      text: columnNames[1],
      style: {
       color: '#3333FF'
      }
     },
    labels: {
     formatter: function() {
      return this.value;
     },
     style: {
      color: '#3333FF'
     }
    }
   }
 });
 // if only one y axis is defined (total: 2 axes)
// addAxis() is not necessary here, since one y axis is already defined
 if (num === 2) {
  highchartPlot.addSeries({
   name: columnNames[0] + " vs. " + columnNames[1],
   color: '#3333FF',
   data: a
  });
 }
// ...
 // add second y axis
 highchartPlot.addAxis({
  // distance to plot frame
  offset: 30,
  title: {
   text: columnNames[2],
   // different color
   style: {
    color: '#006666'
   }
  },
  labels: {
```

```
    // axis labeling
    formatter: function() {
     return this.value;
    },
    // labeling has same color as data points
    style: {
     color: '#006666'
    }
   },
   // true: axis is displayed on the right side of the plot
   // false: left side
   opposite: true
   }, false);
   // define first data (x and y1)
   highchartPlot.addSeries({
    name: columnNames[0] + " vs. " + columnNames[1],
    color: '#3333FF',
    data: y1
   });
   // define first data (x and y2)
   highchartPlot.addSeries({
    name: columnNames[0] + " vs. " + columnNames[2],
    color: '#006666',
    yAxis: 1,
    data: y2
   });
  // ...
}
```

6: Extended function plotHigh()

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die Arbeit keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt habe.

Würzburg, den

Hempfling, Christina