SFB 649 Discussion Paper 2007-041

QuantNet – A Database-Driven Online Repository of Scientific Information

Anton Andriyashin*
Wolfgang Härdle*

* Humboldt-Universität zu Berlin, Germany

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".

http://sfb649.wiwi.hu-berlin.de
ISSN 1860-5664

SFB 649, Humboldt-Universität zu Berlin
Spandauer Straße 1, D-10178 Berlin

QuantNet – A Database-Driven Online Repository of Scientific Information

Anton V. Andriyashin and Wolfgang K. Härdle

CASE – Center for Applied Statistics and Economics
Humboldt-Universität zu Berlin,
Spandauer Straße 1, 10178 Berlin, Germany

Abstract

In this study a framework for an online database-driven repository of information – *QuantNet* – is presented. *QuantNet* is aimed at easing the process of web publishing for those who are unfamiliar with technical details and markup languages. At the same time advanced users are provided with easy user style markup tools while flexible and trouble-free application administration is being a top priority. In this realm a special emphasis is put on the construction of a metalanguage containing only simplest possible structures. Different stages – from low-level text processing via *Atox* to the transformation of XML documents via XSLT, PHP and mySQL – are thoroughly described. The motivation for further possible application extensions like DTD or preliminary document check, based on analytic grammar form, is provided.

JEL classification: C88, C89

Keywords: QuantNet, database-driven, online, repository, XML, XSLT, PHP, mySQL, Atox

Acknowledgements: The financial support from the Deutsche Forschungsgemeinschaft via SFB 649 ”Ökonomisches Risiko”, Humboldt-Universität zu Berlin is gratefully acknowledged.

1 Introduction

1.1 Motivation

Many sociologists consider the XXI century to be the true beginning of the new information era. With the amount of information generated every day and the share of that information being represented in the World Wide Web, it will not be an exaggeration to say that online media become at least as important as their paper-based counterparts.

A substantial portion of scientific knowledge produced is later presented online to share new ideas with colleagues all over the world. Last decade clearly showed the importance of Internet presence, resulting in the emergence of different citation index systems like *ISI*, *Scopus*, *CiteSeer*, *RePeC*, *Google Scholar* and others. But if the aptitude towards online presence is quite clear, it is not always clear *how* to present that information, since substantial technical difficulties can arise while establishing one's own online content system, e.g. in the framework of a research institution.

The aim of this work is to provide a semi-automated core called *QuantNet* allowing to publish significant amounts of scientific information online in the situation when regular updates are implied and, most importantly, when the authors of submitted materials are **not assumed to be aware of any markup language**, i.e. the materials can be submitted as ASCII files with the simplest structure. *QuantNet* is supposed to process that ASCII documents properly and, with the help of different languages like XML, XSLT and PHP, transfer the data into readable, well-formed and consistent HTML structure.

It is not an ultimate goal of this study to provide a ready-to-ship commercial web application. Instead, the implementation of the core, examination of its possibilities and limitations are of particular interest. At the same time, only minor efforts should later be undertaken to deploy a full-scale online system, basing on the created core.

1.2 *QuantNet*: A Look Inside

What is a typical example of a scientific online repository of information? For instance, one could point to a help system of some application like *MATLAB* or *R*, or may be refer to some API description like *MSDN*. Whatever example is taken, the output available to the user is the same – it is a complex set of HTML documents usually with a dynamic navigation control.

Consider a virtual example of a project or a procedure that is about to be submitted online via *QuantNet*. A typical ASCII file could look as follows:

```
@Area SFM
@Name Autocorrelation Plots
@Function_call SFMacfar2()
@Description Plots the autocorrelation function of AR(2) process
@Revision 1.2
@Author Christian Hafner, 2007-01-06

lag=30; lag value
a1=0.5; value of alpha_1
a2=0.4; value of alpha_2
input=readvalue("alpha1"|"alpha2"|"lag", 0.5|0.4|30)
...
```

The fist part of the file contains some general information about the project like its name, author and so on, while the second part may contain a detailed description and/or computer code. As it can be seen from the listing, the ASCII file does not contain any language-specific markup tags. The only tags employed are natural field descriptors, followed by the @ symbol. The author of the submitted document does not have to care about auxiliary properties like font size, color, family and so on. The only thing required is just to follow the sample structure.

QuantNet takes this file and transforms it into a well-formed XML file that separates all important information portions by placing necessary tags. The resulting XML file looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?> <quantlet>
    <name> Autocorrelation Plots </name>
    <area> SFM </area>
    <function_call>SFMacfar2()</function_call>
    <desc>
    Plots the autocorrelation function of AR(2) process
    </desc>
    <rev>1.2</rev>
    <author>Christian Hafner, 20070106</author>
</quantlet>
```

At the same time advanced users of *QuantNet* are supposed to profit from the maximum amount of possibilities offered by native HTML, XML and XSLT, so inline tags, if present, should be processed adequately. For instance, if the <**bold**> tag is an *allowed* one in the ASCII file and stands for the <**b**> HTML counterpart, then *QuantNet* should be able to process the following ASCII file adequately and make this tag nested properly at the later stage.

QuantNet does not limit extra tags only by markup group. In principle even *MathML*, when supported by the browser (e.g. *Mozilla Firefox*) and properly implemented in the master XSLT template – the issue to be discussed in Section 2.2 – should adequately be displayed inside, say, the <**math**> tag. That can be very handy for the documents containing a lot of formulas.

QuantNet is supposed to deliver such a degree of scalability that almost any HTML tag or their combination could later be defined as simpler and more user-friendly tags allowed for input ASCII files.

There exist several solutions that may be helpful in this field, e.g. a lightweight markup language *Textile*, converting simple ASCII files into well-formed XHTML and allowing some formatting variations [4], or *AsciiDoc*, aimed at writing short documents in ASCII to be converted in HTML [1].

These tools can be good at solving some specific web-oriented tasks but are not sufficient to build a complete and scalable content system like *QuantNet*.

In this work the representation part of the content is put solely on XSLT while string manipulation accounts only for the preparation of necessary raw data files in XML. The logic of *Textile*, for instance, is employed exactly at this stage – while creating XML files out of submitted ASCII files – but with one key difference: no style options to appear later inside HTML code are considered at this stage.

The next section focuses on the motivation for employment of XML, XSLT and PHP instead of plain HTML in the situation when a web application consists of numerous smaller independent documents with the same or similar structure. Part 2 describes the major steps of ASCII documents conversion into XML. These XML files constitute *QuantNet* at the later stage – this process is described in Part 3. Finally, in Part 4 several possibilities to expand the possibilities of *QuantNet* even to a greater extent are presented.

1.3 What Is Wrong With Regular HTML Publishing?

A typical application of *QuantNet* could be an online interdisciplinary repository of research materials submitted by various parties – from professional researchers to university students. These materials could contain not only results and algorithm descriptions, which is a traditional form of almost any publication, but also source codes, when available, as well as other supplementary data upon author's wish.

Every publishing entity like a journal has its own styling instructions. The same applies to web publishing. The aim of *QuantNet* is to avoid any of prerequisites that come from a markup field. Instead, *QuantNet* imposes only several restrictions on the original ASCII data files with submitted projects so that each of them contained the author's name, the name of the project etc., refer to Table 1 for more details. And that is all! A researcher should not worry about what font size to employ for a certain heading unless he or she is well aware of specific HTML tags to take advantage of.

In this sense the submitted ASCII files normally are to contain only data and minimum amount (or no) markup tags. This is the fundamental feature of *QuantNet* – a user supplies a structured data file, and *QuantNet* semi-automatically processes this file and incorporates it in the proper cell of the system: the plain data ASCII file becomes a well-formed HTML document with adequate graphic elements and navigation tools.

While it may be clear what advantages provides a submission of a research study as the data ASCII file for a person who is not aware of HTML for online publication, several administration aspects, which may be not so obvious, are worth mentioning here.

Suppose the author of the project to be published online has the file in HTML format. Does that automatically mean that this person is aware of HTML? Not necessarily. Even *Microsoft Word* – one of the mostly used text processor – can save its output as an HTML file [6]. *LaTeX2HTML* is another solution for those preferring LaTeX to *Word*.

So what is wrong with HTML as a submission format or even *Microsoft Word* that can be later converted to HTML? If there is a *single document* to be published, nothing is probably wrong. If there are style and/or document structure prerequisites, they can be matched. However, in the *multiple documents* setup several problems do arise.

Imagine that, for instance, a new version of graphical design of the web application is to be introduced. And if, say, there are 500 HTML documents contained in the system, each of them must be changed one by one! Not to mention the problems of navigation across these individual files and difficulties to introduce content-driven dynamic functions like automatic generation of links to auxiliary materials given, for instance, the project name.

Would not it be greater if the user intending to give a name of the project had to type something like:

```
@Name My First Project in QuantNet
```

instead of caring about all necessary HTML tags that may easily take the following form:

```
<p style="color: red; margin-left: 20px; font: normal oblique 16 px">
My First Project in QuantNet </p>
```

And this is only one element – name. An HTML document with rich formatting contains dozens of such elements. If one of them is to be changed, then all the documents in the system have to be updated.

At what about the navigation? Assuming that HTML frames are not employed following the recommendations of leading web-designers, there is no easy solution in a multiple documents setup for navigation elements.

Fortunately XML, XSLT and PHP could provide a much more efficient solution in these terms. Before addressing these areas in more detail, let us have a closer look at what *QuantNet* gets as an input – an ASCII file with raw project information.

2 Single Document Setup

2.1 Typical Structure of a Submitted ASCII File

Since every XML file normally contains only raw structured information like in a database, the employment of the ASCII files with no markup elements perfectly fits XML in this sense.

The very basic structure of an ASCII file describing, say, some statistical procedure could look as follows. The first file block refers to project cataloging, i.e. it contains relevant information about authors, software platform, project stage an so on. The aim of this substructure is to present *summarized information* about the project in a compact form when it is being viewed by the end-user.

The second block contains the project itself with supplementary computer code for this particular example. Most of information is located at **@desc** field while **@input** and **@output** refer solely to the algorithm implementation.

```
// head block
@project_name     //name of the project
@area             //project area
@short_desc       //short project description
@function_call    //function call for the attached computer procedure
@matlab           //Boolean (yes/no): indicates if procedure is implemented in
  MATLAB
@R                //Boolean (yes/no): indicates if procedure is implemented in R
@author           //author name and contact details
@revision         //project status
// main part
@input            //input arguments for the attached computer procedure
@output           //variables returned by the procedure
@desc             //detailed project description
```

Of course, *QuantNet* should not be limited by exactly that setup, this is just an example of the possibilities of *QuantNet's* core. For instance, social science projects may have no computer code attached, therefore **@input** and **@output** cells are either not filled or just excluded from the master design template. In general such setup can adequately represent the help- or description-based content, consider, for instance, the help system of *MATLAB* as an example of structurally rigorous repository of computer algorithm descriptions.

While XSLT is a style template applied to a given XML file, XML could be generated out of the submitted ASCII file. This important aspect will be regarded later in Section 2.3.

2.2 XML and XSLT – A Single Document in HTML

XSLT is a powerful means to transform structured information from XML into a rich-formatted representation like HTML or even PDF. XSLT is a set of *stylesheet* rules applied to specific portions of XML document and resulting in creation of different style elements that are frequently content-driven. For instance, if the processed portion of information from XML is a heading, then XSLT template may apply the <**h3**> HTML tag.

Let us consider the following code example to realize the architecture of XSLT.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3    <xsl:template match="/">
4
5              <h2 class="padded_text">
6                <xsl:value-of select="quantlet/project_name"/>
7              </h2>
8
9              <!-- RECURSIVE TEMPLATE APPLICATION -->
10             <xsl:apply-templates/>
11             <!-- END OF RECURSIVE TEMPLATE APPLICATION -->
12
13   </xsl:template>
```

The part of the template presented here is the heading of the actual template employed in *QuantNet*. Important is that it has a *scalable structure* – the third line starts to define one global template that matches *all* possible elements of *any* XML file. The final HTML content is mostly generated through a recursive template application <**xsl:apply-templates/**> [7]. These templates are defined after the <**xsl:template match="/"**> element is closed.

For instance, a summary table element (refer to Figure 1), defined in *QuantNet's* XSLT file, looks as follows:

```
<xsl:template match="head">
    <!-- SUMMARY TABLE -->
    <table >
      <div class="box" id="boxContainer">
        <div class="box" id="boxContent">
          <p class="padded_table">
            <xsl:apply-templates/>
          </p>
        </div>
      </div>
    </table>
    <!-- END OF SUMMARY TABLE -->
</xsl:template>
<xsl:template match="head/*">
    <p class="padded_table">
      <b>
        <xsl:value-of select="@print"/>:
      </b>
      <xsl:apply-templates/>
    </p>
</xsl:template>
```

As one can see, this is a very general table definition. Table styles are defined with the help of HTML/CSS, the number of table elements is arbitrary. The <**xsl:template match="head/*"**> tag tells to take *only those* elements of XML file in the table that are nested into the <**head**> tag. In this way one can maintain a truly flexible structure of *QuantNet*, because if later some extra elements are to be added, for instance, to the summary table environment, it would suffice just to ensure the presence of these elements in the XML file in the way the they are properly nested.

Important is that a **single** general XSLT file can be applied to **numerous** XML documents allowing to define one metastyle and render appropriate HTML content for every project file independently.

It is still an open question how to link all the documents together – that issue is to be addressed later in Part 3 of the work. Up to the moment, however, little was said about the ASCII-XML transformation. Having the desired XML and XSLT structure in mind, the proper translation of ASCII files ensures the smooth functioning of *QuantNet* in the single document setup.

2.3 ASCII to XML: *Atox* and XSLT

One of the very first processing steps in *QuantNet* is the transformation of the structured ASCII files into well-formed XML ones. Each submitted ASCII file is supposed to contain the predefined fields or cells, see Table 1 for details.

Field name	Description
@project_name	Project name
@area	Project area, e.g. Economics, Informatics etc.
@short_desc	Project short description, usually one-two sentences
@function_call	Function call of the submitted computer program if any
@matlab	Boolean: yes/no value, indicates if the project is implemented in MATLAB
@R	Boolean: yes/no value, indicates if the project is implemented in R
@author	Project author's name
@revision	Project state, revision number if any
@input	Computer procedure (if any) input variable(s) description
@output	Computer procedure (if any) output variable(s) description
@desc	Full project/computer algorithm description

Table 1: Fields employed by *QuantNet* in ASCII files

By design of *QuantNet*, ASCII files should maintain as easy and natural structure as possible. Therefore, the choice of auxiliary elements like the indication of field start or end should be transparent as well. In this case even an unexperienced user would not have difficulties understanding how to replicate this structure for his/her own project description for online publishing.

How does any text processing unit work looking for a particular element? First of all, it looks for a set of symbols defining the beginning of field. *QuantNet* implies a field every time the symbol @

appears. So, for instance, **@ABCD** would mean that the field **ABCD** is about to begin.

Most certainly, the end of field could be defined analogously, i.e. introducing some extra auxiliary symbol like #. Instead of this, *QuantNet* uses the double new line character as a trigger to an end of field – so there is no need for additional visible character to be introduced. As it is in text, different paragraphs are separated by new lines. Since *QuantNet* is assumed to contain possibly longer text descriptions, a single new line character would not suffice. Therefore, the double new line one seems to be the most appropriate for the case. It is also worth mentioning that if for some reason these symbol choices are not appropriate for the system administrator of *QuantNet*, other *arbitrary* choices could easily be adopted.

Apart from predefined fields, ASCII files for *QuantNet* could carry allowed tags for advanced users. These tags are later translated via XSLT while ASCII-XML conversion should ensure their "survival" in the XML representation. An example of such a construction could be the <**b**> HTML tag – the one creating bold text, refer to Section 4.1 for an example.

When the ASCII file is filled in with content following the aforementioned rules, how does plain text become a well-formed XML document? One common way to implement that is to employ *Yacc/Lex* software, see Section 4.3 for some more details.

However, due to its relative complexity, *QuantNet* does not employ this approach. Instead, a combination of XSLT and *Atox* text processing freeware is used. While *Atox* is capable of translating ASCII data into a limited-form XML, XSLT adds the missing elements and introduces project files as fully capable XML documents that are later transferred into HTML output.

Atox is a freeware Python-based character management tool developed by *Magnus Lie Hetland* [2]. It employs an extra XML markup file with the target patterns to look for in an ASCII file, and these rules are a mix of XML grammar and Python *regular expressions*.

After the first processing iteration is finished, a temporary XML file is created. However, its content lacks the so called XML *attribute values* – the feature that is not directly supported by *Atox* for the reason that XSLT could do the job much more efficiently. Due to this fact, the final ASCII-XML conversion conducts at the second step when an auxiliary XSLT template is applied to a temporary XML file produced by *Atox*.

Therefore, the conversion of submitted project documentation as an ASCII file into a well-formed XML document is performed in two stages. While the first one involves the character management software – *Atox* – the second one ensures the necessary XSLT post-processing.

3 Multiple Documents Setup

3.1 mySQL and PHP

QuantNet, as a system potentially handling numerous documents from different fields, should adequately store and organize this variety of information. The natural solution could be to introduce of one of the database management systems, and **mySQL** is one of them. With its help it would be possible to establish different project categories or areas, maintain hierarchic structures if necessary, and, of course, with a proper scripting language like PHP, implement *QuantNet* as a web application.

While there exist different database management systems along with mySQL, its main advantage is that it is free. Also mySQL is supposed to show really competitive results in terms of data processing speed.

PHP is mainly used as a sever-side scripting language meaning that the server generates the content (usually in HTML form) that is then rendered by the end-user's browser. PHP runs on all of the most popular web servers and is available for many different operating systems. This programming language can be used with a large number of relational database management systems including mySQL.

Field name	Description
ID	Entry number, automatic counter
Name	Entry name to appear in the menu, e.g. *Economics* (area) or
	On Estimation of Additive/Partial Linear Models (project name)
Href	Project XML file name, e.g. *Project01.xml*
Parent	Number of the entry that is parent to the current one,
	0 – if entry has no parents (usually area names)
PublicUse	For internal maintaining purposes only

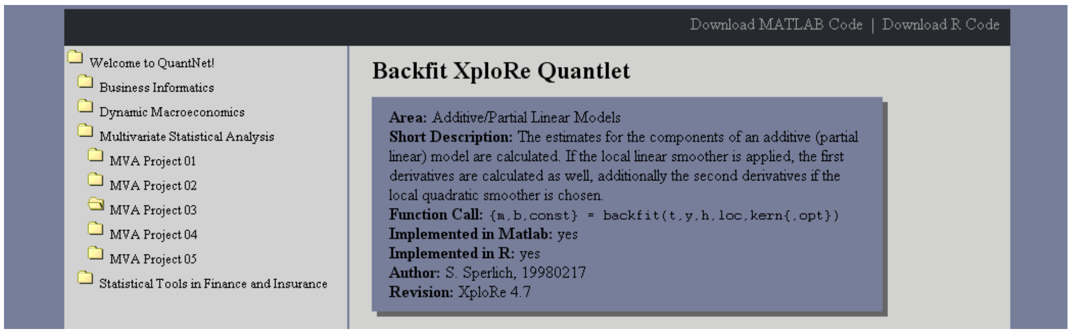
Table 2: Fields employed by *QuantNet* in the mySQL database

If the projects submitted to *QuantNet* are maintained as an hierarchical structure, i.e. each project belongs to a specific area (refer to the **@area** tag in Section 2.3), this setup can be easily replicated in quite a simple database. From there it is possible to build a dynamic multi-level navigation menu, pointing to different project areas (one or more levels) and, at the next level, to project titles. mySQL part of the script was adopted from the free and publicly available *EasyPHPtree* script of *Myiosoft* [3].

So after project files are available in XML format, their names and reference to a particular area are stored in the mySQL database. PHP script generates, among other elements, a dynamic navigation menu that either expands/collapses different folders or loads the appropriate project HTML content, coming as a result of application of the master XSLT template to an XML document, which name becomes known to *QuantNet* by querying the mySQL database that is performed by the PHP script.

Figure 1 provides an actual view of the dynamic navigation menu of *QuantNet* in action.

The right pane of Figure 1 contains the summary table for the project, refer to Figure 3 for the full-page view.



Backfit XploRe Quantlet

Area: Additive/Partial Linear Models
Short Description: The estimates for the components of an additive (partial linear) model are calculated. If the local linear smoother is applied, the first derivatives are calculated as well, additionally the second derivatives if the local quadratic smoother is chosen.
Function Call: `{m,b,const} = backfit(t,y,h,loc,kern{,opt})`
Implemented in Matlab: yes
Implemented in R: yes
Author: S. Sperlich, 19980217
Revision: XploRe 4.7

Figure 1: Dynamic navigation menu of *QuantNet* and summary table environment example

3.2 Putting Everything Together

PHP and mySQL were the last elements needed to be introduced to ensure the smooth functioning of *QuantNet* as a complex web application. In this section let us review the major stages of *QuantNet*.

The two main prerequisites of *QuantNet* are: first, to ease the process of web publishing by introducing a simple and natural metalanguage and second, to make the future application administration as easy as possible. These prerequisites led to the choice of a particular tool to tackle specific challenges. At the very first step, submitted ASCII files are processed by *Atox* and later post-processed by an additional XSLT template. Available field names in ASCII files, extra allowed tags for user formatting and end of field character(s) are three most important factors to influence the work of *Atox*. As a result of these operations, each ASCII file is translated into a well-formed XML document.

The names of these documents (XML files) are stored in a simple mySQL database along with other suitable properties like project affiliation in terms of the area and so on, refer to Table 2 for details. The database provides the administrator not only with an easy way to maintain the logical structure of *QuantNet*, but also it is an important part while the creation of the dynamic navigation menu and possibly other dynamic page elements (e.g. dynamic hyperlinks to auxiliary project files).

PHP script is in charge of building HTML code every time the user clicks on any item in the navigation menu, link or any other dynamic element. Depending on the type of the navigation menu object – it can either be a category name (folder) or the project name – the script loads the appropriate content and expands/collapses the menu. After the user clicks on the menu, to show a particular project, PHP script performs a query to the mySQL database and gets the relevant project document XML file name. Master XSLT template is applied to this XML file and the final HTML content for the current page is created on the server. This code is sent to the client's browser that renders it into the page that user actually views. Figure 2 provides a schematic overview of this process while Figure 3 shows *QuantNet* in action.

4 Possibilities of *QuantNet's* Core

4.1 Scalability – User-defined Tags

While the major features and implementation steps of *QuantNet* have already been described in the previous sections, it is equally important to discuss elements that lead to the true scalability of *QuantNet*.

One of such peculiarities is the ability to add specific tags that are allowed in submitted ASCII files. Consider, for instance, a descriptive part of some project that points to some computer code listing. In this case it would be nice if the author could use the <**listing**> tag, for instance, to indicate this portion of content and apply a suitable font family/size to ease the perception of material.

Another example could be the support of MathML. If a document contains mathematic formulas, they are likely to have been typed either in LaTeX or *Microsoft Word* or any other system. With a large number of converters available nowadays, it would not be a problem to convert them to

MathML and embed in the ASCII file. If the <**math**> tag is the one that indicates the formulas, then the automatically generated MathML code is just to be pasted inside these tags.

And, of course, one should not forget about most basic, though still important, style HTML tags like <**b**>, <**u**> and <**i**> standing for bold, underlined and italic text respectively. Many other analogous examples can be constructed, but important is the following – to implement these features, only one element of *QuantNet* must be changed. And that element is the master XSLT template, which is common for any project XML file in the system.

For instance, if the <**bold**> tag for ASCII files is the one to resemble the <**b**> HTML tag, then the following code should be added to the XSLT template:

```
...
<xsl:template match="bold">
    <b>
      <xsl:apply-templates/>
    </b>
</xsl:template>
...
```

Of course, the implementation of other user tags may not always be so trivial, but the ground principle of *QuantNet* remains intact – any extra tag can later be added to the system by changing only the single file – the common XSLT template – without affecting other *QuantNet's* modules, e.g. markup rules employed by *Atox*.

4.2 Ease of Administration

How does *QuantNet* handle new documents? What happens when a new XML document is to be added to a particular area and incorporated in the navigation menu?

At the first step the new ASCII file is processed by *Atox* and additional XSLT template to get the

XML document – this is done automatically by launching the appropriate scripts. At the second step the administrator should add the new record into mySQL database indicating the name of the XML file and its category by pointing to an appropriate root record that stands for the category name, see Section 3.1 for details. And that is all! The navigation menu is updated automatically by the PHP script, and no extra actions are required from the administrator.

With a freeware tool like *Visual Query Browser*, the administrator can avoid entering time consuming mySQL code to add a new or modify an existing database entry. Instead, visual environment is provided, and even an unexperienced person could easily modify the database.

Hence it will not be an exaggeration to say that *QuantNet* allows almost as smooth and trouble-free system maintaining as possible, assuming that submitted ASCII file follow the proper structure requirements.

4.3 Ways to Make *QuantNet* Even More Powerful

Although this feature is not implemented in the moment, *QuantNet* could potentially perform a preliminary check of submitted files in order to establish if they follow the predefined formatting rules and conversion to XML is successful. Usually this area is associated with so called *XML schemas* and *XML schema languages* that define permitted structure for XML elements and attributes using *regular expressions.*

DTD – **D**ocument **T**ype **D**efinition – is one of the commonly used XML schema languages. DTD is associated with an XML document via so called *Document Type Declaration*, which is a special kind of declaration tag appearing at the beginning of an XML document. For instance, if an XML document is assumed to have a root tag named <**projects_list**> and optionally contain nested tags <**input**>, <**output**>, <**matlab**>, <**R**> and <**function_call**> while the presence of nested tags <**project**> (possibly several instances of that tag), <**area**>, <**author**>, <**short_desc**> and <**desc**> is required (refer to Table 1 for more details), then the following XML DTD describes this setup:

```
<!ELEMENT projects_list (project*)>
<!ELEMENT project (area, author, short_desc, matlab?, R?, function_call?, input?,
   output?, desc)>
<!ELEMENT area (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT short_desc (#PCDATA)>
<!ELEMENT matlab (#PCDATA)>
<!ELEMENT R (#PCDATA)>
<!ELEMENT function_call (#PCDATA)>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT desc (#PCDATA)>
```

In this way XML files can be checked to be compatible with *QuantNet*. However, if a submitted ASCII file contains, for instance, a typo and one of the fields does not match the valid set (refer to Table 1 for details), then *Atox* will not be able to perform a meaningful conversion. Therefore, another way to ensure the smooth work of *QuantNet* is to establish a procedure that pre-checks submitted ASCII files. One possible way to do that is to build a *lexical analyzer* of the ASCII files, for instance, with the help of *Flex* – the software that is available for free. Let us suppose that field names are objects of one's interest. Writing a simple procedure in C, employing *Flex* with a rule that seeks for blocks of characters starting with the @ symbol, will make it possible to get an array of all employed field names. It is only a technical matter then to compare that array with a prespecified set of allowed field names, for example, the one presented in Table 1.

A more sophisticated check would be possible, if one employs, for instance, *Yacc* – the standard parser generator on many UNIX systems. *Yacc* generates a parser (the program part that tries to make syntactic sense of the submitted data) based on an analytic grammar written in a notation similar to *Backus-Naur form* (BNF) [5]. *Yacc* generates the code for the parser in the C programming language. BNF operates with lexemes, and that lexemes are provided by *Flex*, hence *Yacc* is frequently employed together with *Flex*.

If the administrator of *QuantNet* would like to change end of field character in the system, the changes are to be undertaken only in the single file – *Atox* markup XML file containing the rules. Since the current version of *QuantNet* employs *regular expressions* due to the architecture of *Atox*, *QuantNet* is capable to handle almost any possible end of string pattern.

The ultimate goal could be to get a user-uploaded ASCII file and then run a script that processes it, to ensure the syntactical consistency, and converts the file to XML via *Atox* and XSLT, if the file has correct structure. If not, then the script sends an automatically-generated email message, pointing to an error that happened while the syntactical check.

All these described features could be implemented in future, it is just a matter of time.

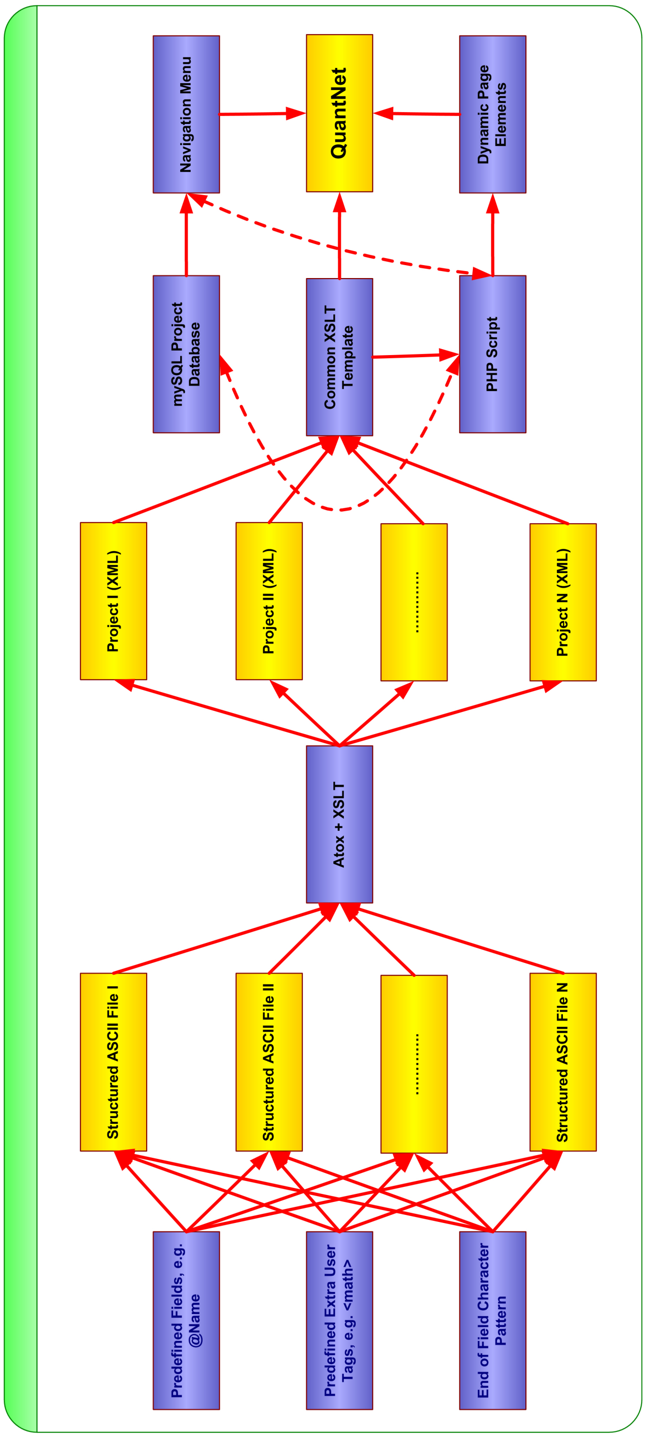
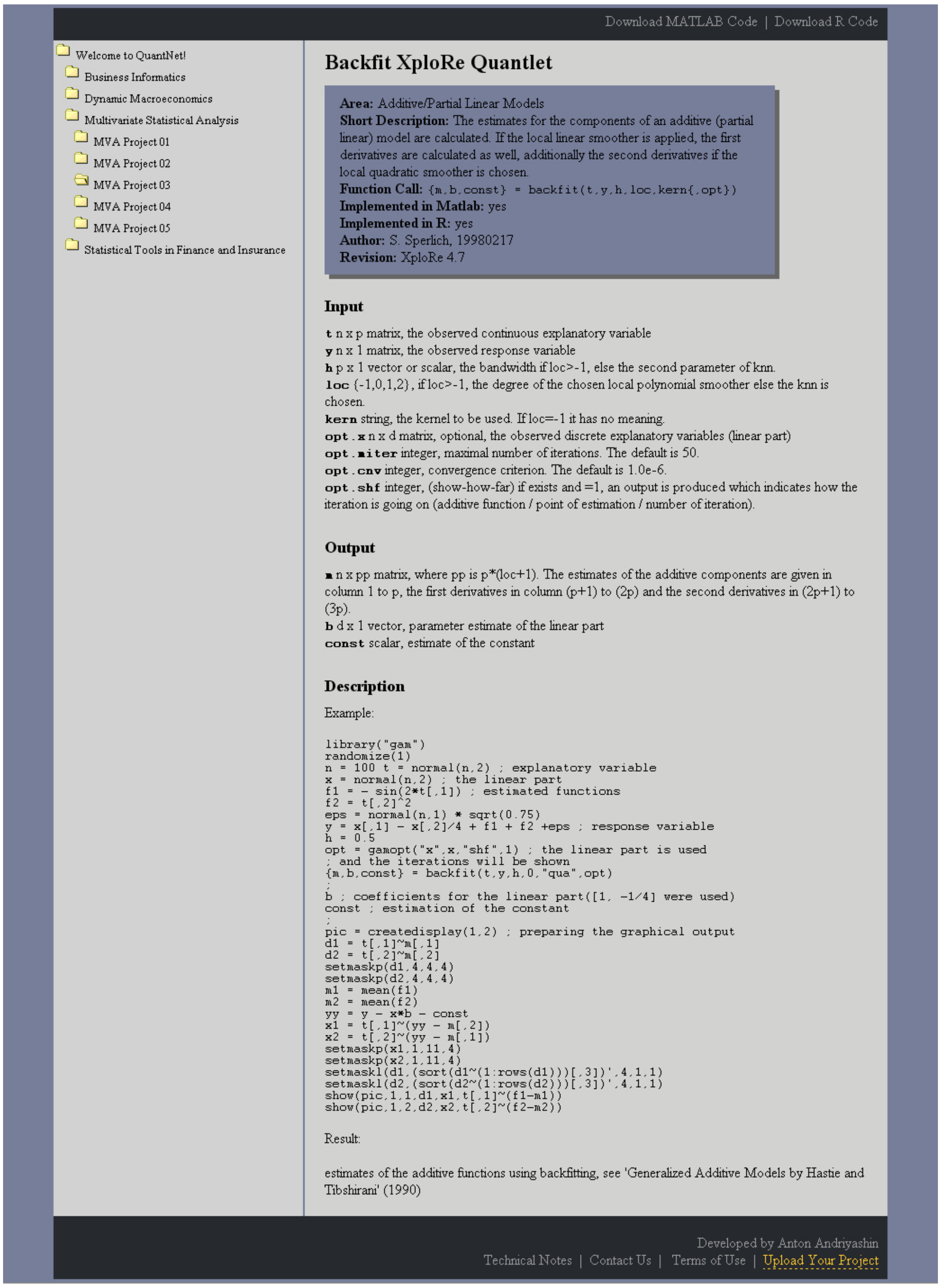


Figure 2: Putting everything together


Download MATLAB Code | Download R Code

Welcome to QuantNet!
Business Informatics
Dynamic Macroeconomics
Multivariate Statistical Analysis
MVA Project 01
MVA Project 02
MVA Project 03
MVA Project 04
MVA Project 05
Statistical Tools in Finance and Insurance

Backfit XploRe Quantlet

Area: Additive/Partial Linear Models
Short Description: The estimates for the components of an additive (partial linear) model are calculated. If the local linear smoother is applied, the first derivatives are calculated as well, additionally the second derivatives if the local quadratic smoother is chosen.
Function Call: `{m,b,const} = backfit(t,y,h,loc,kern{,opt})`
Implemented in Matlab: yes
Implemented in R: yes
Author: S. Sperlich, 19980217
Revision: XploRe 4.7

Input

`t` n x p matrix, the observed continuous explanatory variable
`y` n x 1 matrix, the observed response variable
`h` p x 1 vector or scalar, the bandwidth if loc>-1, else the second parameter of knn.
`loc` {-1,0,1,2}, if loc>-1, the degree of the chosen local polynomial smoother else the knn is chosen.
`kern` string, the kernel to be used. If loc=-1 it has no meaning.
`opt.x` n x d matrix, optional, the observed discrete explanatory variables (linear part)
`opt.miter` integer, maximal number of iterations. The default is 50.
`opt.cnv` integer, convergence criterion. The default is 1.0e-6.
`opt.shf` integer, (show-how-far) if exists and =1, an output is produced which indicates how the iteration is going on (additive function / point of estimation / number of iteration).

Output

`m` n x pp matrix, where pp is p*(loc+1). The estimates of the additive components are given in column 1 to p, the first derivatives in column (p+1) to (2p) and the second derivatives in (2p+1) to (3p).
`b` d x 1 vector, parameter estimate of the linear part
`const` scalar, estimate of the constant

Description

Example:

```
library("gam")
randomize(1)
n = 100 t = normal(n,2) ; explanatory variable
x = normal(n,2) ; the linear part
f1 = - sin(2*t[,1]) ; estimated functions
f2 = t[,2]^2
eps = normal(n,1) * sqrt(0.75)
y = x[,1] - x[,2]/4 + f1 + f2 +eps ; response variable
h = 0.5
opt = gamopt("x",x,"shf",1) ; the linear part is used
; and the iterations will be shown
{m,b,const} = backfit(t,y,h,0,"qua",opt)
;
b ; coefficients for the linear part([1, -1/4] were used)
const ; estimation of the constant
;
pic = createdisplay(1,2) ; preparing the graphical output
d1 = t[,1]~m[,1]
d2 = t[,2]~m[,2]
setmaskp(d1,4,4,4)
setmaskp(d2,4,4,4)
m1 = mean(f1)
m2 = mean(f2)
yy = y - x*b - const
x1 = t[,1]~(yy - m[,2])
x2 = t[,2]~(yy - m[,1])
setmaskp(x1,1,11,4)
setmaskp(x2,1,11,4)
setmaskl(d1,(sort(d1~(1:rows(d1)))[,3])',4,1,1)
setmaskl(d2,(sort(d2~(1:rows(d2)))[,3])',4,1,1)
show(pic,1,1,d1,x1,t[,1]~(f1-m1))
show(pic,1,2,d2,x2,t[,2]~(f2-m2))
```

Result:

estimates of the additive functions using backfitting, see 'Generalized Additive Models by Hastie and Tibshirani' (1990)

Developed by Anton Andriyashin
Technical Notes | Contact Us | Terms of Use | Upload Your Project

Figure 3: The actual look of *QuantNet*

References

[1] *Asciidoc*. http://www.methods.co.nz/asciidoc.

[2] *Atox*. http://atox.sourceforge.net.

[3] *EasyPHPtree*. Publicly available free PHP script at Myiosoft, http://myiosoft.com.

[4] *Textile*. http://www.textism.org/tools/textile.

[5] N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[6] Evan Lenz, Mary McRae, and Simon StLaurent. *Office 2003 XML: Integrating Office with the Rest of the World*. O'Reilly.

[7] Steve Muench. *Building Oracle XML Applications*. O'Reilly, 2000.

SFB 649 Discussion Paper Series 2007

For a complete list of Discussion Papers published by the SFB 649, please visit http://sfb649.wiwi.hu-berlin.de.

001 "Trade Liberalisation, Process and Product Innovation, and Relative Skill Demand" by Sebastian Braun, January 2007.
002 "Robust Risk Management. Accounting for Nonstationarity and Heavy Tails" by Ying Chen and Vladimir Spokoiny, January 2007.
003 "Explaining Asset Prices with External Habits and Wage Rigidities in a DSGE Model." by Harald Uhlig, January 2007.
004 "Volatility and Causality in Asia Pacific Financial Markets" by Enzo Weber, January 2007.
005 "Quantile Sieve Estimates For Time Series" by Jürgen Franke, Jean-Pierre Stockis and Joseph Tadjuidje, February 2007.
006 "Real Origins of the Great Depression: Monopolistic Competition, Union Power, and the American Business Cycle in the 1920s" by Monique Ebell and Albrecht Ritschl, February 2007.
007 "Rules, Discretion or Reputation? Monetary Policies and the Efficiency of Financial Markets in Germany, 14th to 16th Centuries" by Oliver Volckart, February 2007.
008 "Sectoral Transformation, Turbulence, and Labour Market Dynamics in Germany" by Ronald Bachmann and Michael C. Burda, February 2007.
009 "Union Wage Compression in a Right-to-Manage Model" by Thorsten Vogel, February 2007.
010 "On $\sigma-$additive robust representation of convex risk measures for unbounded financial positions in the presence of uncertainty about the market model" by Volker Krätschmer, March 2007.
011 "Media Coverage and Macroeconomic Information Processing" by Alexandra Niessen, March 2007.
012 "Are Correlations Constant Over Time? Application of the CC-TRIG$_t$-test to Return Series from Different Asset Classes." by Matthias Fischer, March 2007.
013 "Uncertain Paternity, Mating Market Failure, and the Institution of Marriage" by Dirk Bethmann and Michael Kvasnicka, March 2007.
014 "What Happened to the Transatlantic Capital Market Relations?" by Enzo Weber, March 2007.
015 "Who Leads Financial Markets?" by Enzo Weber, April 2007.
016 "Fiscal Policy Rules in Practice" by Andreas Thams, April 2007.
017 "Empirical Pricing Kernels and Investor Preferences" by Kai Detlefsen, Wolfgang Härdle and Rouslan Moro, April 2007.
018 "Simultaneous Causality in International Trade" by Enzo Weber, April 2007.
019 "Regional and Outward Economic Integration in South-East Asia" by Enzo Weber, April 2007.
020 "Computational Statistics and Data Visualization" by Antony Unwin, Chun-houh Chen and Wolfgang Härdle, April 2007.
021 "Ideology Without Ideologists" by Lydia Mechtenberg, April 2007.
022 "A Generalized ARFIMA Process with Markov-Switching Fractional Differencing Parameter" by Wen-Jen Tsay and Wolfgang Härdle, April 2007.

SFB 649, Spandauer Straße 1, D-10178 Berlin
http://sfb649.wiwi.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".

SFB 649, Spandauer Straße 1, D-10178 Berlin
http://sfb649.wiwi.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the SFB 649 "Economic Risk".