UPPSALA
UNIVERSITET

UPTEC X 18033
Examensarbete 30 hp
Januari 2019

Implementation of an automatic quality control of derived data files for NONMEM

Eric Sandström

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Master Programme in Molecular Biotechnology Engineering

Eric Sandström

A pharmacometric analysis must be based on correct data to be valid. Source clinical data is rarely ready to be modelled as is, but rather needs to be reprogrammed to fit the format required by the pharmacometric modelling software. The reprogramming steps include selecting the subsets of data relevant for modelling, deriving new information from the source and adjusting units and encoding. Sometimes, the source data may also be flawed, containing vague definitions and missing or confusing values. In either setting, the source data needs to be reprogrammed to remedy this, followed by extensive quality control to capture any errors or inconsistencies produced along the way. The quality control is a lengthy task which is often performed manually, either by the scientists conducting the pharmacometric study or by independent reviewers. This project presents an automatic data quality control with the purpose of aiding the data curation process, as to minimize any potential errors that would otherwise have to be detected by the manual quality control. The automatic quality control is implemented as an R-package and is specifically tailored for the needs of Pharmetheus.

Handledare: Kajsa Harling
Ämnesgranskare: Ola Spjuth
Examinator: Jan Andersson
ISSN: 1401-2138, UPTEC X18033

Sammanfattning

Läkemedelsutveckling är en dyr och långsam process med kostnader på miljardtals dollar (USD) under utvecklingsperioden som ofta kan ta uppåt 20 år. De flesta läkemedel tar sig inte heller ut på marknaden, utan de misslyckas med att uppvisa tillräcklig effekt eller har för många sidoeffekter för att vara värda att utveckla vidare. Kostnaden vid ett misslyckande är därför extremt hög, då vinsten från ett färdigt läkemedel uteblir, och det blir desto dyrare ju senare i utvecklingen det visar sig att läkemedlet inte kommer att fungera. Det finns också etiska aspekter kopplade till de höga kostnaderna, dels kan potentiellt viktiga läkemedel stoppas i utvecklingsfasen av ekonomiska skäl och dels kunde pengar spenderade på ett till slut icke-fungerande läkemedel ha använts på något annat. Således är det viktigt inom läkemedelsbranschen att tidigt fånga de läkemedel som inte kommer fungera, samt att effektivt kunna påvisa den positiva effekten av de som troligen kommer att fungera.

På senare tid har intresset att använda sig av datormodeller för att påvisa effekten av läkemedel ökat. Tanken med datormodellerna är att man ska kunna simulera effekten av läkemedel på vissa patienter, givet att man gjort ett antal kliniska studier på andra patienter först. Om ett läkemedel exempelvis utvecklats och testats på ett antal personer i samband med en klinisk studie, kan datormodellen lista ut hur läkemedlet skulle fungera på andra patienter. På så vis kan man med stor precision förutsäga effekterna av läkemedlet utan att behöva dras med de enorma logistiska kostnaderna som följer av att behöva utföra fler kliniska studier, och som i vissa fall skulle leda till att utveckligen av läkemedlet stoppas. Med datormodellerna kan man även identifiera om vissa läkemedel kan visa sig vara farliga för vissa patienter, till exempel patienter som har nedsatt njur- eller leverfunktion. Datormodellerna tillåter en således också att se vilken dosering en patient bör få av ett läkemedel, baserat på dennes fysiologiska egenskaper (s.k. personlig medicin). I slutändan har datormodellerna alltså flera positiva effekter; de tillåter en att förutse effekterna av läkemedel utan att behöva utföra mängder av dyra och krångliga kliniska studier, de tillåter en att anpassa behandlingar efter individier och de minskar riskerna för patienter eftersom man i förväg kan förutsäga om läkemedlet kan visa sig vara farligt för somliga.

Forskningsfältet som behandlar matematiska datormodeller av läkemedel byggda på klinisk data kallas farmakometri och är ett snabbt växande fält inom läkemedelsutveckling. Skapandet av dessa datormodeller kräver dock klinisk data att utgå ifrån. Det är dessutom ytterst viktigt att den kliniska datan man baserar sina modeller på är korrekt, om det finns fel eller tvetydigheter i datan riskerar modellen att ge fel information som kan få allvarliga konsekvenser. Ett viktigt steg i framtagningen av modellerna är därför att noggrant kurera all klinisk data först. Arbetet som presenteras i denna rapport är en mjukvara med just den uppgiften.

Mjukvaran kallas för autoQualityControl (aQC), engelska för automatisk kvalitétskontroll, och har som mål att granska klinisk data innan den modelleras. Eftersom klinisk data skiljer sig väldigt mycket från projekt till projekt (vilket läkemedel som studerades, vilka patienter som var med o.s.v.), är mjukvaran flexibel och det är användaren själv som bestämmer vilka krav datan måste uppfylla. I princip fungerar alltså mjukvaran på följande vis: användaren matar in den kliniska datan tillsammans med de definitioner som datan förväntas uppfylla. Mjukvaran kollar då först att definitionerna är okej, och därefter att den kliniska datan faktiskt uppfyller definitionerna. Resultatet blir i form av en liten rapport som berättar för användaren om något verkade vara fel. Tanken med projektet är att komplettera den manuella kvalitétskontrollen som utförs på datan, med målet att i förväg identifiera så många potentiella misstag som möjligt, så att kvalitétskontrollsprocessen går så smidigt som möjligt.

Table of contents

Abbreviations

AMT	Dose amount administered
ADaM	Analysis Data Model
WT	Body Weight
CMT	Compartment
CG	Cockcroft-Gault formula
COMP	compliance
CLCR	creatinine clearance
DDT	Data definition table
DIG	digoxin
DIU	diuretics
DV	Dependent Variable
DVO	Original Dependent Variable
EVID	Event Identifer
II	Interdose-interval
Moxo	Moxonidine
NYHA	New York Heart Association
NONMEM	Non-linear mixed effect modeling
PK	Pharmacokinetics
PK-PD	Pharmacokinetics- Pharmacodynamics
PD	Pharmacodynamics
Pheno	Phenobarbital
QC	Quality Control
SCR	Serum Creatinine
SS	Steady-state
SDTM	Study Data Tabulation Model
TAD	Time after previous dose

Introduction

1.1 Pharmacometrics

Paracelsus (1493-1541), credited as the father of toxicology, stated that "All things are poison and not without poison; only the dose makes a thing not a poison". As so, recognizing that the dose-concentration-effect relationship is what distinguishes a therapeutically useful agent from a poison is a fundamental part of clinical pharmacology (Atkinson & Lalonde 2017). The aforementioned relationship is quantitative and thus in nature mathematical, meaning that it is possible to mathematically model these relationships (Atkinson & Lalonde 2017). Pharmacometrics is the branch of science that deals with mathematical models of effects between drugs and patients, based on clinical data. Mathematical models that describe the mechanism of drug-receptor interactions, such as molecular mechanics, is not part of pharmacometrics. Clinical data is data that is obtained from a clinical study, such as information about the patients in the study (body weights, age, etc.) and information about the drug (dosings, measured concentrations etc.). Two broad divisions of pharmacometrics is pharmacokinetic (PK) modelling and pharmacodynamic (PD) modelling. In essence, PK refers to the drugs journey through the body, such as absorption, distribution, metabolism and excretion, whereas PD refers to the body's biological response to the drug. Typically, a pharmacometric model will encompass the results from pharmacokinetic-pharmacodynamic (PKPD) modelling to make inferences on optimal dosing for clinical trials or practice (Standing 2017). That is, by using clinical data, one can find a mathematical model (or several models) that describes what happens to the drug in the body, what the drug does to the body and from there decide how to properly administer the drug to patients. In recent times, pharmacometrics also includes models that take into account other factors, such as disease progression, study compliance and placebo effects (Atkinson & Lalonde 2017).

PD and PK behaviour is expected to vary between people, based on physiological properties such as body weight, age, genotype etc. (Mould & Upton 2012, Standing 2017). A simple example is the effect of alcohol in different individuals, where larger individuals are generally less affected by the same amount of alcohol as compared to smaller individuals, owing to their larger body volume resulting in more dilution of the alcohol. However, size is far from the only factor affecting alcohol tolerance as genetic disposition plays a crucial role as well (Zakhari 2006). Therefore, a model approach that reflects variations in a population is desired, which leads us to the term population modelling. A population model will typically consist of a structural model (algebraic formula or differential equations) to characterize the typical drug behaviour for the population as a whole. Further, physiological properties (covariates) such as age and body weight can help partition the population in sub-populations which may explain some of the variability between subjects. Lastly, all variability cannot usually be explained or predicted, it is for us random, but the magnitude of the unexplained variability exhibited can be estimated (Mould & Upton 2012, Standing 2017). Therefore, a population PKPD model would allow us to identify a model that roughly fits the population, can identify crucial

covariates such as age or weight, and give an indication of the impact of random effects. Note that random effects here refers to variability that cannot be predicted, and might include errors in the data (which might stem from limited measurement accuracy) but also effects that have an unknown underlying cause. Knowing the magnitude of variability for these random effects can be especially important for drugs with a narrow therapeutic window. The therapeutic window of a drug refers to the concentration range between minimum effective concentration and minimum toxic concentration, i.e. concentrations below the minimum effective concentration might have insufficient therapeutic effect whereas concentrations above the minimum toxic concentration are deemed to have adverse side effects.

Although covariates such as body weight or sex certainly can help with classification of patients, and thereby help explain observed variability in their PKPD profiles, there are usually other parameters that might better explain the underlying cause of the variability. Most drugs are eliminated from the body through either hepatic clearance or renal clearance. One would expect a patient with impaired function in either of these organs to eliminate a drug more slowly (Thomas & Thomas 2009), thereby running a risk of higher exposure compared to a normal patient, given that both patients received the same dose of drug. A measure of renal function is creatinine clearance (CRCL), which can be either measured through urine samples or estimated using formulae such as the Cockcroft-Gault (CG) formula (Thomas & Thomas 2009, Dabla 2010). Therefore, CRCL is often included in PK studies as a covariate, as it is often time correlated to the elimination rate of a renally excreted drug. Some care has to be taken if the CRCL was estimated with formulas however, as these are not necessarily accurate for patients with kidney disease, a common symptom in diabetic patients (Dabla 2010). Further covariates of potential interest are concomitant medications. Certain drugs that are metabolised by enzymes in the human body, typically by different members of the Cytochrome P450 family, can reach higher exposures if concomitantly administered with CYP inhibitors (Mohammed *et al*. 2017). Using the same line of thought, if drug elimination is due to metabolism by some enzyme, the genotype of patients for the culprit enzyme might be an interesting covariate to include in a study. Nonetheless, more simple covariates such as age and weight still remain fundamentally important predictors. Renal clearance for instance is typically proportional to weight, age and sex (Thomas & Thomas 2009). Likewise, if a drug is metabolised by a certain enzyme in, for instance, the liver, a larger individual with a larger liver could be expected to metabolise the drug faster, courtesy of having more cells producing the enzyme (Standing 2017).

A pharmacometric study in a PKPD setting aims to characterize the behaviour of a drug and its effect on patients. Identification of the typical exposure-response relationships, which covariates explains the observed variability between patients and which drug doses yields the best effect remains the end goal of most studies. Submissions to regulatory agencies for drug approval saw a steady increase in inclusion of pharmacometric analyses over the years 2000-2008 (Lee *et al.* 2011). The trend is likely to continue. Pharmacometrics studies are carried out using specialized software. The first software developed for such purpose, called Non-

linear Mixed Effect Modelling and abbreviated NONMEM (Beal *et al.* 2014), is still widely used. It is the main analysis software used at Pharmetheus, an Uppsala-based company offering pharmacometric services supporting drug development of compounds in all phases and in all therapeutic areas.

1.2 NONMEM

NONMEM solves pharmacometric problems by optimally fitting the parameters of a mathematical model to observed patient data. In a PK setting, the mathematical model would be a set of equations the pharmacometrician believed would best describe how the concentration of a drug changes over time in a patients body. In the simplest case, say for a renally cleared drug, the parameters for such a model would typically be clearance (amount of blood cleared over time) and volume of distribution (the volume in which the drug exists, e.g. blood volume of a patient). NONMEM will then try to find the values for clearance and volume of distribution which describe the observed patient data, both for the population as a whole and for the individual patients. Population in this case refers to the typical individual among the observed ones, i.e. the average person. Estimates for all individuals are thereafter expressed as deviations from the population estimate where some of the deviation can hopefully be attributed to physiological attributes such as body weight, age or sex, while the remaining deviation will be considered ’unexplained’. Note that ’unexplained’ does not necessarily mean erroneous but rather that the deviation is due to uninvestigated factors, such as genetic factors or measurement and assay imprecision etc.. The output from a NONMEM estimation is thus a measurement of how well the mathematical model fits the observed patient data, optimal parameter estimates for the model and an indication of which covariates were important.

To make NONMEM actually run, two files are required: the NONMEM data file and the NONMEM model file (Beal *et al.* 2014). The NONMEM data file, at Pharmetheus called the derived data file, contains all patient data that NONMEM should model, in the form of a single table. The table consists of a set of columns, which represent the variables, and a set of rows, also called records, which holds all the values. All columns have the same number of rows and there must be no empty values. In addition, there are certain constraints applied to the variables, both by NONMEM itself and by Pharmetheus standards. The structure and contents of the data file will be further elaborated on in the Methods section.

The model file specifies the information required by NONMEM to model what is inside the data file. For instance, the pharmacometric model, definition of parameters and initial parameter estimates are all defined in the model file. The model file will also contain a description of the problem being investigated, the location and file name of the data file, and numerous other options that can be included (Fisher & Shafer 2007 & Beal *et al.* 2014).

Although NONMEM is an old software and quite user unfriendly, it is still considered the go-to software for pharmacometric analysis. This largely owes to historical and practical reasons.

Many pharmacometricians are already proficient in NONMEM and know how to navigate and use the software to its fullest, through years of experience. On top of that, a lot of software tools, such as Perl-Speaks-NONMEM (PsN) (Lindbom *et al.* 2004), were developed entirely to fit analysis by NONMEM.

1.3 Data programming and Quality Control

Data programming is the process of deriving a NONMEM-ready data file based on clinical source data files and is performed by a data programmer.The data definition table (DDT) is the specification of how the NONMEM data file shall be derived; it contains a complete list of variables and their definitions. The purpose of the DDT is twofold, it serves as programming specifications for the data programmer, and as explanation of the data file contents for external reviewers. Once a derived data file has been produced, the DDT and the data file undergo extensive quality control (QC). The QC of the DDT is necessary to ensure that it is properly defined and readable for all parties that will access it, such as pharmacometricians, data programmers and regulatory agencies. The NONMEM data file needs to be rigorously curated as well before it can be modelled by NONMEM. There are three reasons for this. The first reason is that NONMEM requires the data file to have a specific format and variables within need to obey certain rules, and clinical source data is rarely in this format. The second reason is that clinical source data may contain errors, such as missing entries and uncertain values or definitions. The third reason is that a pharmacometric analysis must be based on correct data to be valid. Since NONMEM can be user-unfriendly, with limited graphical representation and sometimes difficult to interpret error messages, careful design of the model file and curation of the data file is necessary (Fisher & Schafer 2007). The quality control of the derived data file is to a large extent manual, a time-consuming process reflecting the complexity of the DDT. Typically, multiple cycles of reviewing and updating are required before the data file can be finalized.

1.4 The Pharmetheus way of working

At Pharmetheus, the process between the start of a project and a finished analysis is an interplay between different parties. A project starts with an order from a customer who wishes for a pharmacometric analysis on some data from a clinical study. A modeller at Pharmetheus will, together with the client, scope the extent of the project and set up the pharmacometric models to be used. The modeller will also create the first iteration of the DDT and pass it on to the data programmer. The data programmer will evaluate the DDT, suggest modifications and this process will be a back-and-forth one between the modeller and the data programmer. Once the DDT has been agreed upon, typically weeks after the first iteration was created, the data programmer will write an R-script that will convert the clinical source data provided by the client into a NONMEM data file. The NONMEM data file, alongside the DDT, are passed on to external QC-reviewers who in turn will return feedback on both the DDT and the

NONMEM data file. In the end, numerous iterations of reviews will have been done before a finalized DDT and data file have been agreed upon.

2 Methods

The purpose of an automatic quality control (automatic QC) is to replace parts of the manual QC to achieve an overall speed-up of the process. In this project, two, somewhat separate, automatic QC processes were developed: a control of the DDT, to ensure that the variables within are properly defined, and a control of the derived data file, to ensure that the actual data is coherent with the specifications in the DDT as well as being physiologically realistic. The automatic QC was developed entirely in R, a programming language for statistical computing and the programming language primarily used at Pharmetheus. The main reasons R is the language of choice at Pharmetheus are the following: 1) It handles large data sets well, 2) It is powerful for visualization of data (through graphs and tables) which is paramount for reporting the results of pharmacometric analyses. The automatic QC was validated using the R package testthat (Wickham 2011), which allows for creation of unit tests. All individual functions in the automatic QC were validated using comprehensive and structured unit tests.

To be able to realize the R-package constituting the automatic QC, certain functionalities had to be developed, including a way to receive the information contained in the DDT and create data structures to hold and operate on this information. Input from the DDT was obtained from an already existing software used at Pharmetheus for creation of DDT:s, called rock. The automatic QC module was also integrated in rock towards the end of the project. The Pharmetheus internal standard for DDT:s is currently being adapted to the clinical research data standards Study Data Tabulation Model, SDTM, (Clinical Data Interchange Standards Consortium 2018) and Analysis Data Model, ADaM, (Clinical Data Interchange Standards Consortium 2018) and thus changed during the course of this project. As a consequence, certain elements of the automatic QC module were not necessarily adopted in the end product. Upcoming sections will describe the concept behind the automatic QC, how it was developed and how it functions.

2.1 Software details

The NONMEM (Beal *et al.* 2014) version used is 7.3 or higher. The software R (R Development Core Team 2007) (version 3.2.3 or higher) was used for development of the automatic QC. R packages testthat (Wickham 2011), stringr (Wickham 2017), dplyr (Wickham *et al.* 2017, dplyr), lubridate (Grolemund & Wickham 2011), rlang (Henry & Wickham 2017) and roxygen2 (Wickham *et al.* 2017, roxygen2) was also used, although packages may be added or deprecated in future updates.

2.2 Overview of the project

The automatic QC module was developed as an R-package containing the necessary functionality to perform its task. The input to the automatic QC module is the DDT and the NONMEM data file which will henceforth be called the derived data file or just data file. In essence, the automatic QC will evaluate the DDT and then evaluate the contents of the derived data file using the DDT and summarise its findings in an error report. A schematic overview of the process can be seen in Figure 1. The derived data file is essentially the data set that should be modelled by NONMEM, existing in a table like structure where the columns represent the different variables and the rows (also called records) contain the values of the variables. The DDT contains the definition of the variables included in the derived data file, what values they may have and any special conditions applicable. The DDT will be supplied by the Pharmetheus tool for DDT creation, called 'rock'. rock is a graphical software tool developed in-house at Pharmetheus for the specific purpose of aiding pharmacometricians and data programmers to create standardized and well-defined DDT:s. The first step of the automatic QC module is thus to verify that the DDT supplied by rock contains no errors. Any errors found will be immediately reported back to rock and the user, in the form of error messages. The process of creating a DDT is often lengthy and requires numerous iterations between pharmacometricians, clients, data programmers and the manual QC. The automatic QC will serve as a means to facilitate this process. After the DDT has been finalized, the data programmer may begin producing the derived data file from the source clinical data. Once the derived data file is finished, the data programmer may pass the derived data file alongside the finalized DDT to the automatic QC module. The output of this step is an error report, containing information regarding anything that seemed amiss within the derived data. The verification of the DDT includes converting the information contained within to a set of R-expressions, which are essentially small pieces of R-code. These R-expressions will then be evaluated on the derived data resulting in a set of TRUEs and FALSEs. Any records in the derived data that evaluated to FALSE had values that were not in accordance with the specifications of the DDT. Upcoming sections will further elaborate on the different components of this process.

2.3 The source data file and the derived data file

The source data file(s) is the data file containing all clinical data received from a client. The source data file will typically be a set of files in SDTM (Clinical Data Interchange Standards Consortium 2018) or ADaM (Clinical Data Interchange Standards Consortium 2018) format with included information such as: Patient Number, Date and Time, Amount of drug given, Measured drug concentration, Body Weight, Sex etc. The source data file is rarely in a format that can readily be modelled by the Pharmetheus modelling software, NONMEM. Instead, the source data needs to be cleaned up, certain variables might have to be redefined and new variables might have to be derived from existing ones to fit the format required by NONMEM. The process of creating the new data file from the source data file is called data

programming and is performed by a data programmer. The new data file is called the derived data file. The rules for what the derived data file should look like is specified in the DDT. Exactly how the source data should be converted to the derived data is not necessarily specified in the DDT but instead relies on the know-how of the data programmer. Table 1 shows a simple example of what source data may look like before and after conversion. Note that the example is purely for illustrative purposes and does not reflect a real dataset. In this example, each variable in the source data had a direct conversion to the derived data, which might not always be the case. For instance, the information about which date the study was performed on was lost in this example. In a real setting, the derived data would likely have contained an additional column specifying the date.

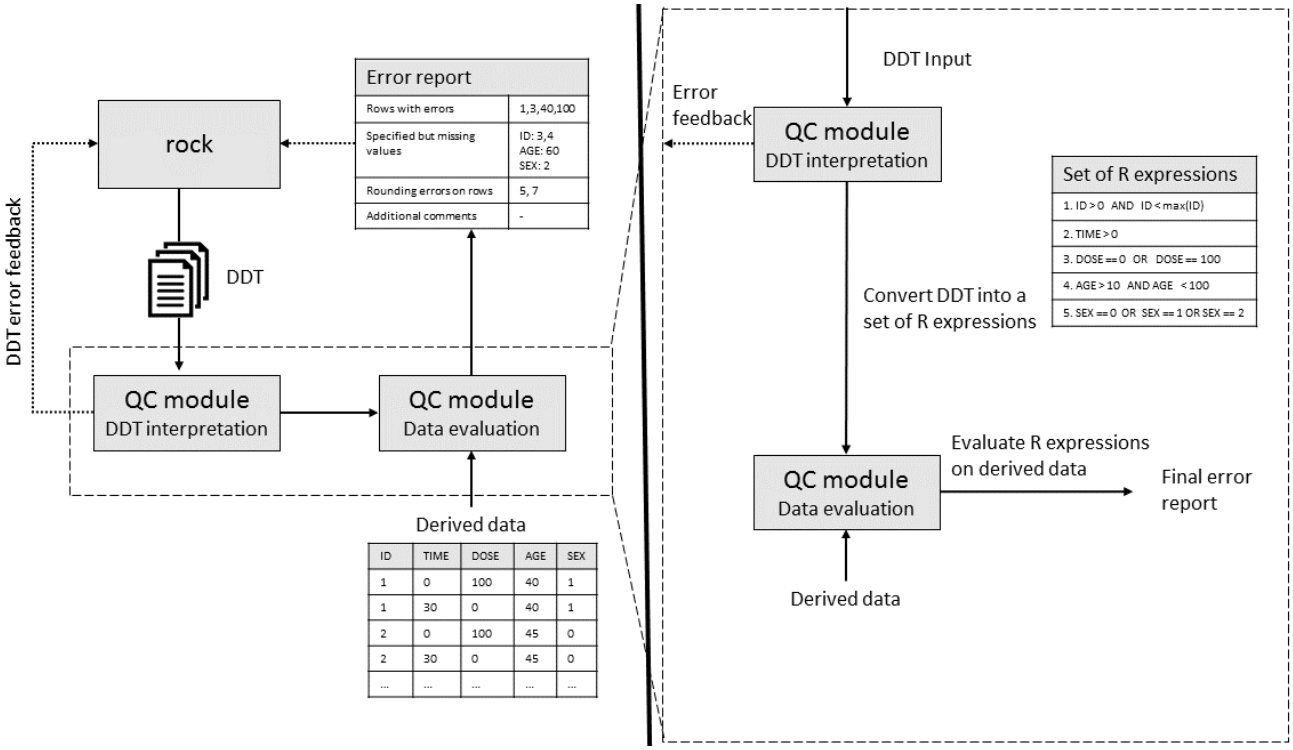

Figure 1. Schematic overview of the automatic QC process. The left-hand side shows the overall process of DDT creation, automatic QC and generation of an error report. The DDT will be supplied to the automatic QC module through the Pharmetheus tool for DDT creation, called rock. Any possible errors within the DDT will be reported back to rock. Once the DDT is verified, the derived data file should be supplied to the automatic QC as well, where it will be evaluated and any errors found will be summarised in an error report. The right-hand side schematically shows the inner workings of the automatic QC module. Given a DDT input from rock, the automatic QC will convert the DDT to a set of R-expressions which are evaluated to TRUE or FALSE when executed on the derived data. Any errors will be summarised in an error report.

2.4 The data definition table

The DDT specifies what information the derived data file should contain. This includes definitions of variables, such as which values they may have, if they need to obey certain rules or number of significant digits. A simplified example of a DDT is shown in Table 2. Although this DDT does not exactly match the dataset presented in Table 1, there are a many similarities, namely the variables ID, TIME, AMT, DV, SEX and WT. Typically, these

variables can be derived more or less directly from the source data, although they may require some processing such as unit conversion. The variable EVID is a NONMEM-specific variable which is never included in SDTM or ADaM data and must instead be produced. Source data that stem from a previous pharmacometric study may however already have EVID defined. EVID, meaning event identifier, is a required variable by NONMEM which tells the software whether a dosing event is happening (i.e. drug was given to patient at the current time point) or if an observation event is happening (drug concentration was measured at the current time

Table 1. Fictional example of source data and derived data. From the source data to the derived data, each variable has had its name and values changed as to fit what was specified in this examples DDT. The new values and names are more in line with what is required by NONMEM.

Source Data

Patient Number	Date and Time	Amount of drug	Measured drug conc.	Weight	Sex
P-001	2001-01-01 06:00	100 mg	0	70 kg	M
P-001	2001-01-01 08:00	0	9 mg/L	70 kg	M
P-001	2001-01-01 12:00	0	6 mg/L	70 kg	M
P2	2001-01-02 7 AM	100 mg	0	90 kg	M
P2	2001-01-02 9 AM	0	8 mg/L	90 kg	M
P2	2001-01-02 1 PM	0	4 mg/L	90 kg	M
P03	2001-01-02 07:00	100 mg	0	60 kg	F
P03	2001-01-02 09:00	0	13 mg/L	60 kg	F
P03	2001-01-02 13:00	0	10 mg/L	60 kg	F

Derived Data

ID	TIME	AMT	DV	WT	SEX
1	0	100	0	70	0
1	120	0	9	70	0
1	240	0	6	70	0
2	0	100	0	90	0
2	120	0	8	90	0
2	240	0	4	90	0
3	0	100	0	60	1
3	120	0	13	60	1
3	240	0	10	60	1

point) and NONMEM will behave accordingly. However, for NONMEM to actually behave accordingly, certain variables, such as AMT, RATE and DV in this example, must follow certain rules. As can be seen in Table 2, which values AMT, RATE and DV can have depends on the current value of EVID. When EVID is 0, AMT and RATE must also be 0 while DV must be non-zero. If a drug dose was given to a patient (EVID = 1) and a measurement (EVID = 0) was made simultaneously, this would have to be expressed on two separate records in the data file, one where EVID is 1 and the other where EVID is 0. Nonetheless, the TIME variable can still be identical for both records, i.e. NONMEM allows multiple events to happen simultaneously but they need to be distinguished using EVID on separate records.

As previously mentioned, the variable EVID is unlikely to already be present in clinical source data meaning that it is the data programmers job to construct the derived data set in a way that includes EVID in a correct manner. For the simple example shown in Table 1, this

would probably mean creating a new column called EVID and set its value to 0 wherever AMT is 0 and set its value to 1 where AMT is 100. NONMEM requires records to be ordered ascending by first ID, then TIME and then EVID. This ordering is not specified in the DDT. If the data is not ordered, NONMEM will crash or give the wrong results. On a final note, the DDT shown here is grossly simplified, containing only seven variables. DDT:s in real projects could easily contain 100 variables with far more complex descriptions and dependencies on other variables. For instance, a single study may contain multiple different drugs, patient groups, placebo groups and so on.

Table 2. Example DDT for explanatory purposes. Each variable in the DDT is represented by its name, a set of possible values, and a description of it. Included in the description is the dependencies it may have on other variables. The example shown here is typical for what a Pharmetheus DDT would look like when presented in a report.

Variable	Possible Values	Description	Unit
ID	1 to number of patients	Unique patient identifier	N/A
TIME	≥ 0	Time stamp for record. Rounded to 2 decimals.	hours
EVID	0, 1	Event identifier. 0 = observation event 1 = dose event	N/A
AMT	0, > 0	Amount of drug given **Records where EVID = 0:** 0, No drug dose given during observation events > 0, For dosing events, dose amount administered at TIME	mg
RATE	0, -2	Rate of infusion **Records where EVID = 0 and AMT = 0:** 0, Infusion rate is zero when no dose is given **Records where EVID = 1 and AMT > 0:** -2, The negative value indicates a special case that will be handled accordingly by NONMEM	N/A
DV	0, > 0	Measured drug concentration (dependent variable) **Records where EVID = 0:** > 0, observed drug concentration at current time point. Rounded to 2 decimals. **Records where EVID = 1:** 0, drug concentration is not measured during dosing events	mg/L
WT	≥ 0, -99	Body weight of patient at baseline. Must be constant across all records for a patient. -99 = Missing value.	kg
SEX	0, 1, -99	Sex of patient. Must be constant across all records for a patient. 0 = Male, 1 = Female, -99 = Missing value.	N/A

2.5 ‘rock’, an in-house software for DDT creation

At Pharmetheus, the DDT:s are created using a software called ’rock’, which was developed in-house at Pharmetheus and is managed by the technical solutions team. The input to the DDT is entered through rock’s graphical interface. Figure 2 shows a part of the rock interface, namely the part necessary for defining a variable that should be included in the DDT. The fields enclosed by red rectangles are arguably the most important ones for properly defining a variable. For instance, the definition of the variable AMT seen in Table 2 is the result of automatic translation of the input seen in Figure 2. All red-boxed fields in Figure 2 takes free text as input. However, the input is still expected to follow some rules for most of the fields, in order to be interpretable by the automatic QC module. For instance, the Dependencies on other variables (4) field must contain an expression in pure R-code, contain at least one variable name and not contain certain illegal characters. The Possible Values (5) field is less strict but will only successfully parse certain input. The Rounding to Decimals (7) field must be blank or contain an atomic number. The Definition (2) and Description (6) fields are meant to be read by humans only and the input to those, at least for the purpose of the automatic QC module, may contain any text. One final detail worth mentioning regards if multiple possible values (5) should have the same dependencies (4). If so, the dependency should only be written on the same row as the first possible value, and the dependency field should be left blank for the other possible values for which the dependency condition applies. For instance, in the example shown in Figure 2, if ’-99’ was also a possible value with the dependency ’EVID == 0’, the value -99 (typically denotes a missing value) would be entered on the third row with the dependency field for that row left blank. rock would interpret this as: ’carry over’ the last explicitly defined dependency and use it for this possible value, i.e. use ’EVID == 0’ for -99.

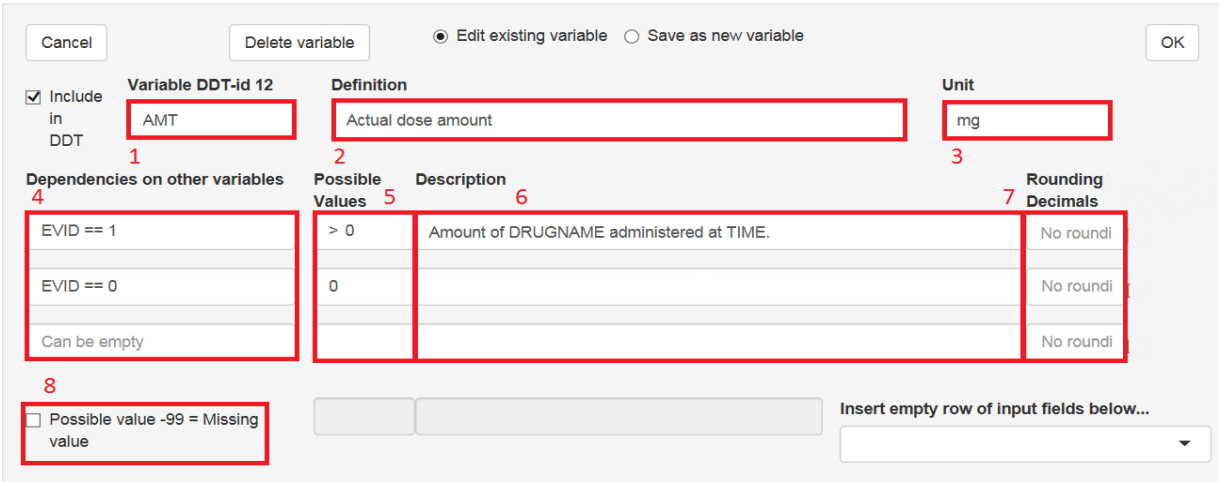

Figure 2. Part of the rock interface for creation of variables for the DDT. 1. The variable name. 2. Brief definition of the variable. 3. Expected unit for the variable. 4. Possible dependencies the variable may have on other variables, interpreted in this example as: AMT must be greater than zero if EVID is 1 and AMT must be zero when EVID is 0. 5. Possible values the variable may have. 6. Description of the variable for human readers. 7. How many decimals the values should be rounded to. 8. Option to include -99 to indicate missing values in the data. This is typically applicable if certain information about a patient is uncertain from the source data. *The figure is not an exact image of the interface but has been slightly altered for visibility purposes.

3 Results

3.1 Interpretation of the DDT in R

To perform automatic QC on the DDT and the derived data, the DDT must somehow be imported into the automatic QC module and the user input interpreted. In brief, the interpretation is the process of converting the user input which comes as text (as seen in Figure 2), into valid R-expressions. As the DDT is created in rock, the input from rock will be fed into the automatic QC module either directly or after the user takes some specific action, such as pressing a button. Referencing back to the image of the rock interface, Figure 2, all fields enclosed by red rectangles are not relevant for the automatic QC module. More specifically, field 2, ’Definition’, and field 6, ’Description’, should contain human-readable text meant to be read by humans only (or any sufficiently intelligent being) and are therefore ignored by the automatic QC module. Remaining fields, for each variable, will be compiled into a data frame, a built-in table-like data structure in R, and the data frame will in turn be the actual input to the automatic QC module.

The example DDT in Table 2 would in R look as in Figure 3. Note the absence of any Definition or Description column. The Variable, Unit, Dependencies, PossibleValues and Decimals columns in Figure 3 are direct translations of fields 1, 3, 4, 5 and 7 respectively in Figure 2. The column ValueNotInData tells the automatic QC module whether the possible value (when combined with its respective dependency) on the same row is expected to be in the derived data. For instance, for the DDT in Figure 3 and for variable WT, the derived data must contain at least one patient whose body weight is missing in the source data (indicated by -99). The column Vartype contains a number which indicates possible special conditions for the variable such as if the value must be constant for a patient (e.g. SEX) or non-decreasing for a patient (e.g. TIME). Currently, the number 7 denotes that a value must be constant within a patient, the number 8 that the value must be non-decreasing and the number 9 that there is no special condition. The columns ValueNotInData and Vartype are also specified in rock although this is not shown in Figure 2.

	Variable	Dependencies	PossibleValues	Decimals	Vartype	ValueNotInData	Unit
1	ID	*NA*	1 to number of patients	0	9	FALSE	*NA*
2	TIME	*NA*	>= 0	2	8	FALSE	hours
3	EVID	*NA*	0	0	9	FALSE	*NA*
4	EVID	*NA*	1	0	9	FALSE	*NA*
5	AMT	EVID == 0	0	0	9	FALSE	mg
6	AMT	EVID == 1	>0	0	9	FALSE	mg
7	RATE	EVID == 0 & AMT == 0	0	0	9	FALSE	*NA*
8	RATE	EVID == 1 & AMT > 0	-2	0	9	FALSE	*NA*
9	DV	EVID == 0	0	0	9	FALSE	mg/L
10	DV	EVID == 1	>0	2	9	FALSE	mg/L
11	WT	*NA*	>=0	0	7	FALSE	kg
12	WT	*NA*	-99	0	7	FALSE	kg
13	SEX	*NA*	0	0	7	FALSE	*NA*
14	SEX	*NA*	1	0	7	FALSE	*NA*
15	SEX	*NA*	2	0	7	FALSE	*NA*
16	SEX	*NA*	-99	0	7	FALSE	*NA*

Figure 3. Representation of the DDT in Table 2 when imported into R. The columns Variable, Unit, Dependencies, PossibleValues and Decimals are direct translations of the fields 1, 3, 4, 5 and 7 shown in Figure 2. The ValueNotInData column specifies whether that specific possible value should be present in the derived data or not. The Vartype column contains a number which specifies if there are any special conditions for the variable in the derived data. 9 means no special condition, 8 means that the value must be non-decreasing for a patient (typical for variables TIME and AGE), and 7 means that the value must be constant within a patient (typical for variable SEX or baseline covariates).

The DDT in Figure 3 has merely been imported by the automatic QC but no actual interpretation of it has been made. In essence, the interpretation step will try and interpret each row of the input DDT. If successful, a new DDT will be generated, with additional columns containing interpreted input. Any fields that cannot be interpreted will generate an error message. Most columns in the input DDT will not be interpreted, but rather, their content will be copied directly to the new DDT. The columns for which interpretation actually happens are the Dependencies and PossibleValues columns. The basic procedure for interpreting the DDT in pseudo-code looks as in Algorithm 1. The result is essentially a copy of the old DDT with two new columns added, one containing the interpreted dependencies and one containing the interpreted possible values. Some alterations may be made to the Dependencies column based on the ’carry over’ principle of dependencies discussed in Section 2.5. Note that the contents in the Dependencies and PossibleValues column are considered as text strings and interpretation of these are made using concepts of string pattern searching using regular expressions (R Development Core Team 2007, Grolemund & Wickham 2017) , see Appendix 1 for more information.

Algorithm 1. Procedure for interpreting a DDT. The interpretation is only performed for two columns in the input DDT, namely the Dependency and PossibleValues column. The other columns are kept as-is in the new DDT.

```
1. Receive input DDT from rock.
2. Copy original DDT and use as template for new DDT
3. Iterate over all rows in the DDT, grouped by variables in the Variable
column
          3.1 Run a pre-check of Dependencies column for current variable.
          The pre-check will return only warnings if it finds any issues.
          Also carries-over dependencies to rows below when applicable

          3.2 Run interpretation of Dependencies column for current
          variable. The result is a new, interpreted Dependency column
          with values as follows:
             - If the input is blank, return TRUE
             - If the input is uninterpretable or erroneous, return FALSE
             - If none of the above, return a copy of the input

          3.3 Run interpretation of PossibleValues column for current
          variable. The result is a new, interpreted PossibleValues column
          with values as follows:
             - Check if input is in a pre-defined expression library:
                    - if no, return FALSE
                    - if yes, return TRUE

          3.4 Append results from 3.2 and 3.3 as new columns in the DDT.

4. Return the new, interpreted, DDT
```

The algorithm presented in Algorithm 1 treats the interpretation of the Dependencies column and PossibleValues column differently. The reasons for this is because the automatic QC expects the input in the Dependencies column to already be R-expressions, whereas it does not expect the input in the PossibleValues column to be. Thus, the interpretation for the Dependencies column is a check that the inputs are correct R-expressions, with some additional rules. The input in the PossibleValues column, on the other hand, is not expected to be proper R-expressions. Instead, the automatic QC will make a look-up in a pre-defined list of recognised expressions and if the input in the PossibleValues exists in the list, the automatic QC will convert it to an R-expression. Note that whenever interpretation fails, the resulting R-expression will be a logical FALSE. Figure 4 shows the interpreted version of the DDT from Figure 3. Two new columns have appeared, namely the ’dep’ and the ’posValue’ columns which contain the interpreted dependencies and possible values respectively. Here, the interpretation was successful throughout as can be seen by the absence of any FALSEs in either column. Note that the TRUEs in the ’dep’ column are due to the blank fields in the original Dependencies column. All fields in the ’dep’ and ’posValue’ column contains R-expressions that can be executed. All columns other than the ’dep’ and ’posValue’ column are identical to those in Figure 3.

The most important part of the interpretation process is that the results are R-expressions which can be readily evaluated. In fact, for control of the derived data, the automatic QC will execute these R-expressions, which in turn will be evaluated to either TRUE or FALSE depending on the actual values of the variables in the derived data file. In principle, the content of the 'dep' column and the 'posValue' column will be joined together to form the full expression that should be evaluated on the dataset. This concept is further elaborated on in Section 3.2. There are some types of inputs to the PossibleValues column that cannot be directly converted to R-expressions. How these types of inputs are handled is discussed in Section 3.1.2.

	Variable	Dependencies	PossibleValues	Decimals	Vartype	ValueNotInData	Unit	dep	posValue
1	ID	*NA*	1 to number of patients	0	9	FALSE	*NA*	TRUE	ID>= 1 & ID <= max(ID)
2	TIME	*NA*	>= 0	2	8	FALSE	hours	TRUE	TIME >= 0
3	EVID	*NA*	0	0	9	FALSE	*NA*	TRUE	EVID == 0
4	EVID	*NA*	1	0	9	FALSE	*NA*	TRUE	EVID == 1
5	AMT	EVID == 0	0	0	9	FALSE	mg	EVID == 0	AMT == 0
6	AMT	EVID == 1	>0	0	9	FALSE	mg	EVID == 1	AMT > 0
7	RATE	EVID == 0 & AMT == 0	0	0	9	FALSE	*NA*	EVID == 0 & AMT == 0	RATE == 0
8	RATE	EVID == 1 & AMT > 0	-2	0	9	FALSE	*NA*	EVID == 1 & AMT > 0	RATE == -2
9	DV	EVID == 0	0	0	9	FALSE	mg/L	EVID == 0	DV == 0
10	DV	EVID == 1	>0	2	9	FALSE	mg/L	EVID == 1	DV > 0
11	WT	*NA*	>=0	0	7	FALSE	kg	TRUE	WT >= 0
12	WT	*NA*	-99	0	7	FALSE	kg	TRUE	WT == -99
13	SEX	*NA*	0	0	7	FALSE	*NA*	TRUE	SEX == 0
14	SEX	*NA*	1	0	7	FALSE	*NA*	TRUE	SEX == 1
15	SEX	*NA*	2	0	7	FALSE	*NA*	TRUE	SEX == 2
16	SEX	*NA*	-99	0	7	FALSE	*NA*	TRUE	SEX == -99

Figure 4. Interpreted version of the DDT in Table 2. The DDT is very similar to that in Figure 3. The difference is the addition of two columns, dep and posValue, which contains the interpreted input of the columns Dependencies and PossibleValues respectively. All entries in the dep and posValue columns are R-expressions that can be executed.

3.1.1 Uninterpretable and erroneous input to the DDT

Certain input to the automatic QC will not be interpretable and the automatic QC will respond with error messages. What is considered uninterpretable or erroneous input to the automatic QC is stipulated in Algorithm 2. The first condition in Algorithm 2 does not have any direct input on the interpretation of the DDT but will return warnings to the user. Input that fulfil conditions listed in 2 and 3, will however cause the automatic QC to convert the input to logical FALSEs, as well as returning error messages. The reason that input fulfilling conditions listed in 2 and 3 converts the input to logical FALSEs is to prevent runtime errors from happening further downstream in the execution. That is, input that gets converted to logical FALSEs is input that may otherwise cause the automatic QC to crash or yield unexpected results. Also, the conversion to logical FALSEs (recall that a logical FALSE is a perfectly viable R-expression that can be executed), ensures that the automatic QC will still run, although it will always evaluate to FALSE, and thus signal to the user that the automatic QC was still performed but there were errors in the input. Input fulfilling conditions 2.1 and 2.2 in Algorithm 2 have to be converted to logical FALSEs as these would otherwise guarantee the code to crash later on. Condition 2.1 essentially says that the input was not an

R-expression at all, which then can not be executed and condition 2.2 says that the input was an expression that contained no or an unknown variable, which essentially translates to an expression being evaluated on non-existing variables. If an input dependency fulfils condition 2.3, it will not necessarily cause any crashes but the expression would be nonsensical. The characters ’=’ and ’<-’ both denote assignment in the R-language so an expression:

```
> EVID = 0
```
or

```
> EVID <- 0
```

would mean ’assign the value 0 to the variable EVID’. The purpose of the automatic QC is to execute R-expressions that evaluates to either TRUE or FALSE. Thus, assignment operations are forbidden. The character sequences ’&&’ and ’||’ are forbidden as they have a special meaning in the R-language. In many programming languages, these character sequences denotes the logical AND operator and logical OR operator respectively. This is true for the R-language as well, but use of double ampersand or vertical slash in R means that the expression is no longer an vector operation. Rather, for vector operations, a single ampersand or vertical slash should be used. The importance of vector operations will be discussed in Section 3.2.

Regarding the input in the PossibleValues column, input that fulfils condition 3.1 will be converted to a logical FALSE. Interpretable input to the automatic QC for the PossibleValues column are defined by a set of pre-defined string pattern matching expressions. If the input matched a pre-defined expression, it will be converted to a working R-expression, based on the recognised input. If not, it will be converted to a logical FALSE. See Appendix 2 for the pre-defined expressions.

Finally, the conditions 1.1-1.3 in Listing 2 operates only on the Dependencies column but will not do any interpretation or conversion of the input. Rather, input that fulfils these conditions will generate only warnings to the user. The reason why these checks exist is twofold. The first reason is to catch potential mistakes that would occur otherwise and the second is to adhere to standards implemented in rock, the tool for DDT creation at Pharmetheus. Condition 1.1 is enforced as it prevents certain nonsensical input from occurring. 1.2 is enforced as it both prevents certain nonsensical input from occurring, but is also necessary for standards in rock. 1.3 is enforced exclusively to adhere to rock standards. The standard in question relates to how dependencies are expected to be ’carried over’ in the rock graphical interface, as explained in 2.5.

Algorithm 2. The Automatic QC will consider the input in the DDT erroneous if it fulfils any of the conditions presented here. Conditions listed in 1. Pre-check of dependencies warnings, are merely warnings and will not have a direct impact on the interpretation. Input fulfilling conditions listed in 2 and 3 will cause the interpreted result to be a logical FALSE.

```
1. Pre-check of dependencies warnings:
          1.1 If a variable has only one explicit dependency. An explicit
          dependency is anything that is not a blank field.
          1.2 If the first dependency for a variable is a blank field, but
          is followed by an explicit dependency.
          1.3 If a variable have duplicated dependencies, apart from blank
          fields.

2. Interpretation of the dependencies:
          2.1 If the dependency is not a valid R-expression.
          2.2 If the dependency contains no variable name or an
          unrecognised variable name.
          2.3 If the dependency contains illegal character sequences which
          are the following:
                    &&, ||, =, <-

3. Interpretation of the possible values:
          3.1 If the possible value was NOT in the pre-defined glossary
          list of viable inputs.
```

Figure 5 showcases a few examples of uninterpretable input. The fields enclosed by red boxes highlights which rows would return errors or warnings in the interpretation step. Figure 5a and Figure 5b shows the same DDT with Figure 5a showing the dependency errors and Figure 5b the possible values errors. In Figure 5a, the first error, for variable AMT, appears because it violated condition 1.1 in Algorithm 2. The dependency reads EVID == 1 and is the only dependency for variable AMT. This means that the variable would be undefined for any value of EVID other than 1. The variable EVID in NONMEM datasets will always take on more values than the value 1, typically 0 and 1 are used. Note that the 'dep' column for this row does not read FALSE however. This is because the automatic QC will still allow this type of input to pass, albeit with a strong warning. The second error, for variable RATE, appears because the dependencies violates conditions 2.3 in Algorithm 2. For the keen eye, one might believe that the dependencies for variable DV violates condition 1.3, as dependencies

EVID == 0 and EVID == 1 are duplicated. However, in this example, the duplicates are a result of the 'carry over' principle that happens in step 3.1 in Algorithm 1. Thus, the user input did not contain duplicated dependencies but they were rather produced by the automatic QC.

In Figure 5b, only one red box is present, namely for the variable EVID. Here the possible values read 'obs. event' and 'dose event'. By Pharmetheus and NONMEM standards, the variable EVID should typically take on the values 0 or 1 (although 2, 3 and 4) are sometimes used too). Recall that EVID is the variable used for event identification, where EVID = 0 actually indicates an observation event (e.g. drug level was measured in a patient) and EVID

= 1 indicates a dosing event (e.g. a certain dose of drug was administered to a patient). Nonetheless, the automatic QC will not accept input that reads 'obs. event' or 'dose event' and instead raise an error. As described in Algorithm 1, whenever an interpretation of a field in the PossibleValues column fail, the corresponding field in the posValue column will become a logical FALSE.

	Variable	Dependencies	dep
1	ID	*NA*	TRUE
2	TIME	*NA*	TRUE
3	EVID	*NA*	TRUE
4	EVID	*NA*	TRUE
5	AMT	EVID == 1	EVID == 1
6	RATE	EVID == 0 && AMT == 0	FALSE
7	RATE	EVID == 1 && AMT > 0	FALSE
8	DV	EVID == 0	EVID == 0
9	DV	EVID == 0	EVID == 0
10	DV	EVID == 1	EVID == 1
11	DV	EVID == 1	EVID == 1
12	WT	*NA*	TRUE
13	WT	*NA*	TRUE
14	SEX	*NA*	TRUE
15	SEX	*NA*	TRUE
16	SEX	*NA*	TRUE
17	SEX	*NA*	TRUE

(a). Errors or warnings appeared for three rows during interpretation of the dependencies. Row 5, for the variable AMT, raised a warning because only a single dependency, 'EVID == 1', was present for this variable. Effectively, this would mean that AMT is undefined for any value of EVID that is not 1. The automatic QC does not explicitly forbid this however but it does return a warning. Rows 6 and 7 had interpretation errors for the variable RATE. The errors rose because of the double ampersand (&&) in the dependencies expressions. In R, and especially for the automatic QC module, single ampersand should be used. Because of this, the dep column for these rows reads FALSE.

	Variable	PossibleValues	posValue
1	ID	1 to number of patients	ID>= 1 & ID <= max(ID)
2	TIME	>= 0	TIME >= 0
3	EVID	obs. event	FALSE
4	EVID	dose event	FALSE
5	AMT	>0	AMT > 0
6	RATE	0	RATE == 0
7	RATE	-2	RATE == -2
8	DV	0	DV == 0
9	DV	-99	DV == -99
10	DV	>0	DV > 0
11	DV	-99	DV == -99
12	WT	>=0	WT >= 0
13	WT	-99	WT == -99
14	SEX	0	SEX == 0
15	SEX	1	SEX == 1
16	SEX	2	SEX == 2
17	SEX	-99	SEX == -99

(b). Errors appeared for rows 3 and 4 for the variable EVID. The automatic QC module can not interpret the text 'obs. event' and 'dose event'. Therefore, the posValue column reads FALSE for these rows. Proper input in this particular case would have been 0 and 1 respectively, as these are the values that EVID should typically have by Pharmetheus and NONMEM standards. Any interpretation error in the PossibleValues column will always result in the corresponding row in the posValue column to become FALSE.

Figure 5. DDT with errors or warnings that appeared during the interpretation step. Rows for which errors or warnings appeared are enclosed by red boxes. For (a), only the Variable, Dependencies and dep columns are shown. For (b), only the Variable, PossibleValues and posValue columns are shown.

3.1.2 Interpretation of special cases

Certain variables will have acceptable inputs to the PossibleValues column that cannot be easily converted to R-expressions. If such an input is encountered, the automatic QC will insert a key word in the posValue column instead of producing an R-expression. The key word in turn, will signal to the automatic QC that it needs to treat these variables in a special manner. There are three inputs that triggers this special case handling, listed below:

- The input is interpreted as a date or a clock time. The input has a format similar to YYYY-MM-DD or HH:MM:SS. The key word 'QCDATE' will be appended to the input as a suffix, e.g. 'YY-MM-DD_QCDATE'.
- The input reads 'character' (not case sensitive). The key word 'QC_CHAR' will be inserted into the posValue column.
- The input reads 'numerical', 'numeric' or 'number' (not case sensitive). The key word 'QC_NUMBER' will be inserted into the posValue column.

Exemplified in R, this process would produce a DDT as seen in Figure 6. Rows 2 and 3 are by the automatic QC considered to be dates or clock times and therefore have the suffix '_QCDATE' appended. The prefixes, HH:MM and YYYY-MM-DD, are kept to tell the automatic QC what format is expected, i.e. in this example, the variable DATE would consider 2018-01-01 an acceptable value but not 18-01-01. Row 8 had 'numerical' as input which was interpreted as 'QC_NUMBER'. 'QC_NUMBER' is a vague definition that accepts anything that is an actual number. Therefore, using numerical, number or similar as input is accepted but not recommended. Row 10 had 'character' which translates to 'QC_CHAR'. 'QC_CHAR' is extremely vague as nearly everything can be interpreted as a character. Historically at Pharmetheus, the input 'character' has been used as possible value for variables denoting dates. Therefore, if a variable whose possible value was interpreted as 'QC_CHAR' also has a name resembling a date (such as DATE or DATETIME), the automatic QC will attempt to evaluate the variable as a date.

	Variable	PossibleValues	posValue
1	ID	1 to number of patients	ID>= 1 & ID <= max(ID)
2	CTIM	HH:MM	HH:MM_QCDATE
3	DATE	YYYY-MM-DD	YYYY-MM-DD_QCDATE
4	EVID	0	EVID == 0
5	EVID	1	EVID == 1
6	AMT	0	AMT == 0
7	AMT	>0	AMT > 0
8	WT	numerical	QC_NUMBER
9	WT	-99	WT == -99
10	ORIGID	character	QC_CHAR

Figure 6. DDT with input that requires special attention, namely for rows 2, 3, 8 and 10. For rows 2 and 3, the automatic QC module has interpreted the input as being dates (clock times qualifies as dates) and therefore appended '_QCDATE' to the posValue column. Row 8 has been interpreted as a number, any number, and is therefore flagged as a 'QC_NUMBER'. Row 10 has been interpreted as a 'character' and is therefore flagged as a 'QC_CHAR'.

3.2 Evaluation of the derived data file

The main task of the automatic QC module is to evaluate whether the derived data file fulfils the requirements specified in the DDT. To do so, the automatic QC needs an interpreted DDT for which the process of creation has been described in previous sections. The derived data file is supplied by the user and evaluated without any modifications done to it. Recall that the interpreted DDT contains a set of R-expressions to be evaluated on the derived data. The R-expressions to be evaluated are essentially the dependencies combined with the possible values. Consider the dep and posValue columns in Figure 4 for variables AMT and EVID. The variable AMT has one possible value that is 'AMT == 0' and a corresponding dependency that reads 'EVID == 0' (row 5). The full R-expression for AMT here would be:

```
> (EVID == 0 & AMT == 0)
```

This expression would evaluate to TRUE when both EVID and AMT in the data are 0 and to FALSE for all other cases. The second dependency and possible value combination for AMT, from row 6 in Figure 4, would be:

```
> (EVID == 1 & AMT > 0)
```

Thus, AMT is defined by two R-expressions. The R-expressions for EVID would look as follows:

```
> (TRUE & EVID == 0)
> (TRUE & EVID == 1)
```

The 'TRUE' component of the expressions above will always evaluate to TRUE, meaning that the expressions effectively only checks whether EVID is 0 or 1. Recall that the TRUE component of the expressions stem from the Dependencies column being blank. Figure 7 contains two tables, with the left table holding 12 values for EVID and AMT respectively. The table to the right contains the aforementioned R-expressions and the result of evaluating these R-expressions on the values for EVID and AMT. For each row and for each variable, one would expect that at least one (preferably no more than one) of the fields reads TRUE. For a variable, if all fields for a row reads FALSE, this means that the corresponding row in the data contained a value that did not fulfil the DDT specifications. Such is the case for AMT on row 12 in Figure 7, which owes to EVID being 0 and AMT 100 for that row in the derived data table. If the automatic QC module evaluates the derived data file, and finds a row for which all fields for a certain variable were FALSE, the erroneous row will be added to the error report.

The automatic QC takes advantage of R's built-in vectorised functionality. Most functions in R are vectorised, meaning that the functions can operate on whole vectors at once, in contrast to having to loop through the vector and running the function on each element individually. This offers a huge runtime speed-up in R, where looping structures are generally slow whereas vector operations are heavily optimized. The leftmost table in Figure 7 can be considered to be composed of two vectors, namely EVID and AMT. The R-expressions presented in the rightmost table are executed as vectorised functions, thus, the results seen on rows 1-12 are the product of evaluating each expression once. As datasets grow larger, and

NONMEM data sets can easily contain hundreds of thousands of records, utilizing the vectorised approach is crucial to achieve manageable runtimes.

	Derived data		EVID		AMT	
	EVID	AMT	TRUE & EVID == 0	TRUE & EVID == 1	EVID == 0 & AMT == 0	EVID == 1 & AMT > 0
1	1	200	FALSE	TRUE	FALSE	TRUE
2	0	0	TRUE	FALSE	TRUE	FALSE
3	0	0	TRUE	FALSE	TRUE	FALSE
4	0	0	TRUE	FALSE	TRUE	FALSE
5	1	150	FALSE	TRUE	FALSE	TRUE
6	0	0	TRUE	FALSE	TRUE	FALSE
7	0	0	TRUE	FALSE	TRUE	FALSE
8	0	0	TRUE	FALSE	TRUE	FALSE
9	1	100	FALSE	TRUE	FALSE	TRUE
10	0	0	TRUE	FALSE	TRUE	FALSE
11	0	0	TRUE	FALSE	TRUE	FALSE
12	0	100	TRUE	FALSE	FALSE	FALSE
			Errors Found: None		Errors Found: row 12	

Figure 7. Example of a derived data set (left table) and illustrated output of the automatic QC module (right table). The second row of the right table contains the R-expressions used to evaluate the derived data. For each row and for each variable, at least one field must read TRUE. If all fields read FALSE, it indicates that the corresponding row in the data contained a value that was not defined in the DDT. In this example, row 12 for AMT contained only FALSEs indicating that the derived data was erroneous for that row.

3.2.1 Decimals and special conditions

The evaluation performed in the previous section checks if the variables in the data fulfils the possible values and dependencies defined for them in the DDT. It does not however check if the number of decimals are correct (if applicable) or if any so-called special conditions are met. For clarification regarding the decimals, consider the variable DV in Figure 4. DV has one possible value definition that reads `DV > 0`. The value 100 qualifies for this definition, and would pass the control in the previous section, but since the DDT also specifies that DV should be rounded to two decimals, `DV = 100` would not pass the decimal check. Instead, for it to pass the decimal check, the value needs to be `DV = 100.00`

Regarding the special conditions, there currently exist two. These are identified by the number in the Vartype column as seen in Figure 4. 7 means ’constant within subject/patient’, 8 means ’non-decreasing within subject/patient’ and 9 means no special condition. A subject or patient is defined by the ID variable that must be present in every dataset (spare possibly if the study was conducted on only one patient). ID is supposed to be ordered increasingly, starting at 1 and ending at the number of patients involved in the study. The derived data set should also be ordered by ID such that all records for patient number 1 are grouped together, followed by the records for patient 2 etc. Typically, variables such as WT (body weight at

baseline) and SEX will have the requirement to be constant for a patient. Variables such as AGE and TIME will typically have the requirement to be non-decreasing for a patient.

3.2.2 Physiologically realistic values

Certain variables are commonly reoccurring in different studies. These are typically covariates such as body weight, age, height, serum creatinine and creatinine clearance. If present, the automatic QC module will attempt to evaluate whether these variables have physiologically realistic values, after having evaluated the data file using the DDT. What defines a physiologically realistic value depends on the variable investigated, what unit it is measured in and also the study setting. Here, study setting might refer to what organism the study was conducted in, whether the patients in the study had certain conditions heavily altering their values and so on. The study setting is unfortunately not passed to the automatic QC. Therefore, the automatic QC will assume that the study was conducted in human adults. If a common physiological variable is present in the dataset, the automatic QC will see if it can understand the unit the variable is expressed in. If so, the automatic QC will use a predetermined range and evaluate if the values of the variable is within this range. For the used pre-determined ranges, see Appendix 3. The ranges used are based on feedback by QC reviewers, which in turn are based on data from CDC (2016). In addition, the ranges used are also based on what has been observed from datasets.

The need of a control like this depends on how the DDT is defined. For instance, the variable AGE (age of patients) might in the DDT be defined as a range from the youngest patient to the oldest patient. In this case, the regular evaluation of the data file using the DDT would detect if any patient was not inside that range. However, more commonly, the variable AGE would be defined simply as ’larger than zero’, which would unfortunately lead to unrealistically old ages passing, should such values have slipped into the data. Therefore, the control of physiologically realistic values serves as a backup, to catch any rogue values, should the definition in the DDT be somewhat lax.

3.3 Test run on moxonidine dataset

The automatic QC module was tested on a simulated dataset from a pharmacometric study on moxonidine (Karlsson *et al.* 1998). The dataset itself was acquired from the Perl-speaks-NONMEM (PsN) (Lindbom *et al.* 2004) repository (Nordgren & Freiberga, 2018) whereas information about the dataset was drawn from the publication (Karlsson *et al.* 1998). In brief, the study was a Phase II dose-finding study of oral moxonidine tablets against placebo in people with congestive heart failure. Patients were placed in one of four groups: the placebo group or one of three moxonidine dose groups. Active treatment started at 0.1 mg twice daily and was then changed to a predefined dose of 0.1, 0.2 or 0.3 mg twice daily. Sampling of patient drug concentration was performed at two occasions, after the first dose and after 12 weeks of therapy. 7 measurements were conducted at each occasion, targeted between 0.5 and 8 hours after the morning dose. Covariates considered during the study were: age, weight, New York Heart Association classification (NYHA), creatinine clearance (CLCR), serum

creatinine (SCR), dose group, actual dose, occasion, gender, study compliance, and concomitant medication. To the best of my knowledge, the dataset used here only contained data from non-placebo patients, totalling 74 patients and 1166 records. Certain variables that existed in the dataset, but was not mentioned in the article, where omitted. These variables were possibly created by the authors for testing purposes and might not have relevance for the final study described in (Karlsson *et al.* 1998). As no proper DDT was available for the dataset, one was created based on the information available in the article and the dataset (Table 3). Unsurprisingly, the automatic QC module reported back some errors after evaluating the dataset using the DDT in Table 3. After all, constructing a DDT based solely on the article and a glimpse of the dataset, without being part of the study, is not obvious. Therefore, no deeper analysis of the errors found was made. Rather, this example showcases what the automatic QC module can be expected to detect when applied to a real dataset. Remember, however, that for a real projected conducted at Pharmetheus, the users of the automatic QC module will be those working with the DDT and dataset, thus analysis of any errors found will be possible. With this in mind, one assumption will be made regarding the errors found by the automatic QC in the moxonidine example. The assumption is that the errors reported can have one of two explanations: 1) Erroneous assumptions about the definitions of variables when creating the DDT were made or 2) There were actual errors in the data. The errors found were the following:

```
Variable DVO: record 503 did not match any Possible Value plus Dependency
condition
Variable SCR: records 735-743, 752-760 and 829-836 did not match any
PossibleValue
Variable DIG: was not constant for patient with ID = 324
Variable DIU: was not constant for patient with ID = 224
```

Starting with variables DIG and DIU, these (together with variable ACE) indicate concomitant medication with digoxin, diuretics and ACE inhibitors respectively. Considering that each patient in the study received moxonidine treatment at two different occasions, it is fully possible that they either began or stopped concomitant medication in between the moxonidine treatment occasions. Upon closer inspection of the dataset, this appeared to be the case as the variables DIG, DIU and ACE were all constant for a patient on each visit. Thus, the errors detected for DIG and DIU were likely due to erroneous assumptions made by me upon creation of the DDT. Noteworthy however is that all patients except for patients 224 and 324 had constant concomitant medication over the entire course of the moxonidine study.

Regarding the variable SCR, which denotes serum creatinine levels, the information was directly drawn from (Karlsson *et al.* 1998). That is, the Possible values definition in Table 3, 60 to 170 umol/L is based on the minimum and maximum values as reported in Table 1 in (Karlsson *et al.* 1998). Interestingly, the records 735-743, 752-760 and 829-836 reported back by the automatic QC all had SCR values higher than the reported maximum, i.e. 170 umol/L. Namely, these records had the following values: 190, 180 and 180 respectively, for patients with IDs: 801, 802 and 901 respectively. The reason for these values surpassing the maximum

described in the article is unknown to me although it is possible that these records where omitted from the final model but left in the dataset which I had access too.

Finally, regarding the variable DVO, the value for DVO on record 503 was 0.0322 ng/mL. Since the variable EVID was 1 for this record, the automatic QC expected a value > 100 ng/mL as seen in Table 3. All other records for which EVID was 1, DVO had values ranging from 110 to 1120 ng/mL (mean 416 ng/mL). Therefore, the value of DVO for record 503 is certainly an outlier although it is not certain that it is an error. Interestingly, the value of DVO for record 504 is also 0.0322 ng/mL indicating the possibility that the value on record 504 was used for record 503, perhaps due to the original value on record 503 missing.

On a final note, the variable ID should typically for a Pharmetheus project be numbered consecutively from 1 to the number of patients involved in the study. In the moxonidine dataset, the first ID was 110 and increased all the way to 1407 with different incremental steps, as there were only 74 different patients involved in the study. Currently, the automatic QC does not enforce consecutive numbering starting from 1, but it does however raise an error if an ID shows up twice if not in direct succession. Since no error was raised for the variable ID in this dataset, the condition was met, although this numbering would probably not have been acceptable in a Pharmetheus project.

Table 3. DDT for the moxonidine dataset. Definitions for the variables are derived from available information in the study article as well as from shallow analysis of the dataset.

Variable	Possible values	Description	Unit
ID	110 to number of patients	Unique patient identifier	N/A
VISI	1, 2	Session for patient	N/A
DGRP	7, 8, 9	Which dosing group a patient belongs to. Must be constant for a patient.	N/A
DOSE	200, 400, 600	Total daily dose amount	ug
FLAG	0, 1	Help identifier for EVID. FLAG = 0 is a dose or reset-dose event. FLAG = 1 is an observation event.	N/A
DVO	> 0, > 100	Original dependent variable (i.e. measured drug concentration). Rounded to 4 decimals. **Records where EVID = 1:** > 100, expecting a higher drug concentration directly after dose **Records where EVID = 0 or 4:** > 0, expecting a lower drug concentration after some time	ng/mL
SCR	60 to 170	Serum creatinine level	micromol/L
AGE	43 to 78	Age of patient. Must be non-decreasing for a patient.	Years

SEX	1, 2	Sex of patient. Must be constant for a patient. 1 = male, 2 = female	N/A
NYHA	1 to 4	New York Heart Association classification. Higher value means more heart complications.	N/A
WT	41 to 125	Body weight for patient	kg
COMP	0, 1	Compliance to study regime. 0 = Not compliant, 1 = Compliant	N/A
ACE	0, 1	Concomitant medication with ACE inhibitors. Must be constant for a patient. 0 = No, 1 = Yes	N/A
DIG	0, 1	Concomitant medication with digoxin. Must be constant for a patient. 0 = No, 1 = Yes	N/A
DIU	0, 1	Concomitant medication with diuretics. Must be constant for a patient. 0 = No, 1 = Yes	N/A
TAD	0 to 10	Time after previous dose	hours
TIME	≥ 0	Time stamp for record. Should be non-decreasing for a patient. Rounded to 2 decimals	hours
CLCR	30 to 142	Creatinine clearance. Calculated with Cockroach-Gauss formula.	mL/min
AMT	100, 200, 300	Actual dose administered. **Records where EVID = 0:** 0, no drug dose given during observation events. **Records where EVID = 1:** 100, for regular dosing events. **Records where EVID = 4:** 100, 200 or 300, for reset-dose events.	ug
SS	0, 1	If the dosing is a steady-state dose. **Records where EVID = 4:** 1, the dose is a steady-state dose. **Records where EVID = 0 or 1:** 0, the dose is not a steady-state dose.	N/A
II	0, 12	Inter-dose interval, gives the time between implied doses. **Records where EVID = 4:** 12, interval dosing. **Records where EVID = 0 or 1:** 0, no interval dosing.	hours
CMT	1, 2	Which compartment is being used. **Records where EVID = 0:** 2, the observation compartment. **Records where EVID = 1, 4**: 1, the dosing compartment.	N/A
EVID	0, 1, 4	Event identifier. **Records where FLAG = 1:** 0, observation event **Records where FLAG = 0:** 1, dose event. 4, reset-dose event.	N/A

3.3.1 Introducing errors in the moxonidine dataset

For further testing of the automatic QC, a script was made to insert new values on random variables in the moxonidine dataset on random records. These new values could possibly lead to errors in the data. Before the script was run, the errors detected in the original data, described in the previous section, were removed as to not cause confusion with this test. The alterations made and their predicted effects are listed in Table 4. Here, all predicted errors were discovered by the automatic QC module. The script was run an additional 100 times, and in all cases, the automatic QC module discovered any errors produced.

Table 4. Changes made to the dataset and which records these changes were made on. Included are the predicted errors of these changes which the automatic QC should pick up on.

Variable & record	Description	Predicted error
FLAG, 1151	Changed value from 1 to 0	No error
EVID, 743	Changed value from 0 to 4	EVID error on record 743 EVID error on record 1151 (due to FLAG change)
DVO, 322	Appended three zeros to decimals	DVO decimal error on record 322
TIME, 781	Changed value from 2.50 to 0	TIME decimal error on record 781 TIME will no longer be non-decreasing for corresponding patient (ID = 805)
NYHA, 691	Changed value from 3 to 0	NYHA error on record 691
SEX, 137	Changed value from 1 to 2	SEX will no longer be constant for patient on corresponding record (ID = 128)
DOSE, 820	Changed value from 200 to 1000	DOSE error on record 820
AGE, 98	Changed value from 73 to 78	AGE will no longer be non-decreasing for patient on corresponding record (ID = 126), assuming the change was not made on the last record of patient.
SS, 826	Changed value from 0 to 1	SS error on record 826 SS error on record 743 (due to EVID change)
II, -	No change	II error on record 743 (due to EVID change)
AMT, -	No change	AMT error on record 743 (due to EVID change)
CMT, -	No change	CMT error on record 743 (due to EVID change)

3.4 Runtime of the automatic Quality Control

The runtime for parsing the DDT is near instantaneous and is likely not going to be an issue for reasonably sized DDT:s. The runtime for evaluating the data file is expected to scale roughly linearly with the size of both the DDT and the data file. What constitutes the size of

the DDT is not only the number of variables included but also the number of definitions for each variable, as interpreted by the automatic QC. For instance, in Figure 6, the variables EVID, AMT and WT all have two definitions each, thus totalling 10 variable definitions for 7 different variables. Figure 8 shows a graph of runtimes for three different datasets, namely Phenobarbital, Moxonidine and Gamma. Moxonidine is the same dataset that was discussed earlier in the result section. Phenobarbital is a dataset that was acquired from the PsN repository (Nordgren & Freiberga 2018) and is based on a study conducted in 1985 by Grasela and Donn. Gamma is a dataset that was generated for testing purposes. Although it is composed of real-like definitions of variables, it is not related to any published study. The sizes of the DDT:s for the datasets were as follows, Phenobarbital; 8 variables with 12 definitions, Moxonidine; 32 variables with 52 definitions, Gamma; 40 variables with 91 definitions. The original sizes for the data files of each data sets were as follows, Phenobarbital; 744 records, Moxonidine; 1166 records, Gamma; 100 records. The contents of the data files were then duplicated numerous times for the purpose of generating large datasets for runtime evaluation. As can be seen in Figure 8, datasets with below 100 000 records should typically run in less than 10 seconds, albeit depending on the users machine. As the number of records grow above 200 000 records, the number of definitions in the DDT start to have an impact as well. Data sets containing up to a million records are rare at Pharmetheus, thus, runtimes for the automatic QC should seldom exceed two minutes.

The underlying complexity of the variable definitions have an impact on runtime as well. Recall that variable definitions are evaluated as R-expressions by the automatic QC. Thus, if the R-expression is complex, which could be due to multiple dependencies on other variables, the evaluation will be slower than that for a simple R-expression. Two R-expressions are presented below, where the first one is expected to run slower than the second one:

```
[1] EVID == 0 & (TYPE == 1 | TYPE == 2) & BLQ == 0 & DV > 0
[2] TRUE & EVID == 0
```

The meaning of the variables included in the expressions is not important here, but the point is rather that the first expression will run slower than the second one as it has to evaluate four variables whereas the second expression only has to evaluate one. Thus, the total runtime of the automatic QC will be dependent on the complexity of the variable definitions as well.

Finally, the runtime will depend on the users machine. For this evaluation, a machine with the following processor was used: Intel Core i7-5600, 2.6 GHz.

3.5 Validation of the automatic quality control

Validation of the automatic QC was done using the R-package testthat (Wickham 2011). Testthat was used to create unit tests, which in turn were organized in testing suites. The purpose of the unit tests is to ensure that each individual function in the automatic QC returns exactly what it is supposed to. If the code of a function would later change, the unit tests allows the developer to verify that the function still does what it should, and serves as additional documentation for new developers joining the project. Organizing the unit tests in

testing suites further allows users and developers to easily execute all tests at once, to get an comprehensive overview of anything that might be amiss with the code, or verify that it still functions as expected.

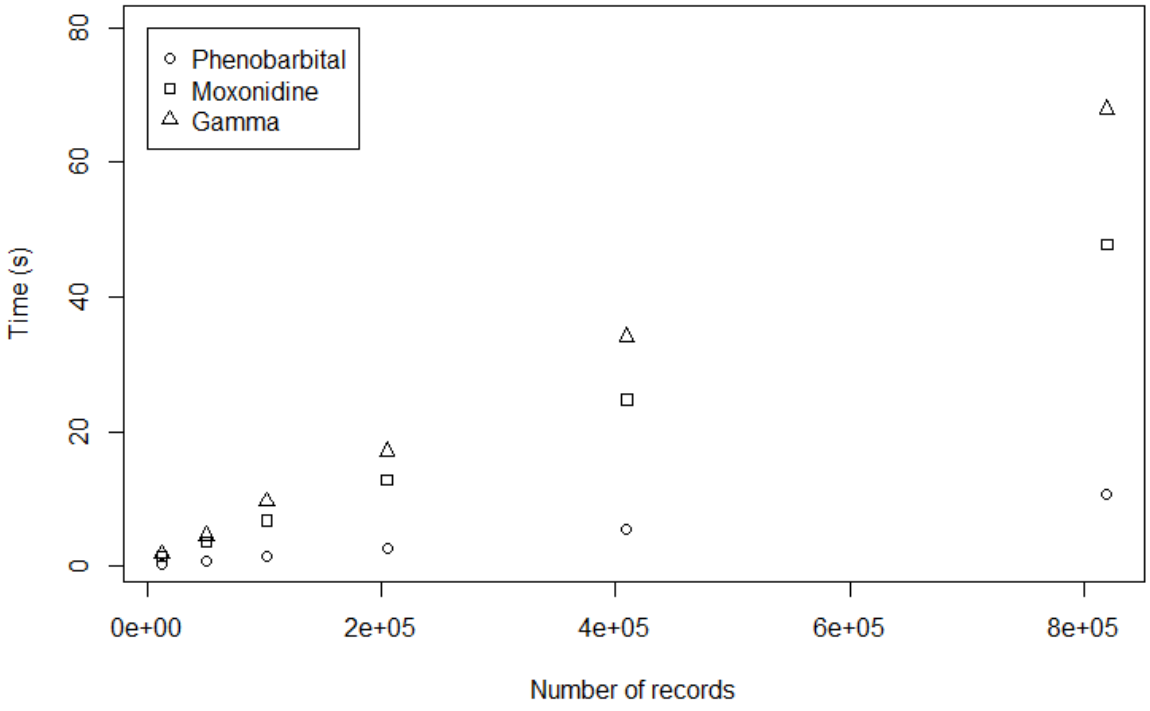

Figure 8. Runtime for evaluations of data files plotted against the number of records in the data files. The original data files did not have the indicated number of records but were rather duplicated numerous times. The legend reflects datasets containing DDT:s of different sizes, with Phenobarbital having 12 variable definitions, Moxonidine having 52 variable definitions and Gamma having 91 variable definitions. The processor of the computer used was the following: Intel Core i7-5600, 2.6 GHz.

4 Discussion

Over the course of this project, an automatic QC was developed aimed at controlling that the DDT and the derived data file for NONMEM fulfilled necessary specifications. While the project came to an end, there are certain functionalities and features (or lack thereof) that deserves further discussion. This section will discuss the reasoning behind the implementation of certain features and which improvements could be made for future updates.

4.1 Sanity check of the DDT

The current implementation of the automatic QC module does some interpretation of the DDT as has been described in Section 2. The automatic QC module checks that it can understand

the input and work with it but it does not check whether the logic is actually sound. Logic in this case refers to the dependencies a variable might have on other variables. Consider the quite simple DDT shown in Figure 9, where the Dependencies and PossibleValues column contain perfectly acceptable input to the automatic QC module. Here, no errors or warnings would be raised. However, if one considers the dependencies and possible values for the variables AMT and RATE, on rows 3 to 6, one realises that the logic is broken. The dependency for RATE on row 6 requires EVID to be 0 and AMT to be greater than 0. However, the dependency on row 3 tells us that AMT must be 0 when EVID is also 0. Therefore, the dependency for AMT on row 3 and the dependency for RATE on row 6 cannot both be true. Depending on how the actual data looks, the automatic QC module would likely detect this error when evaluating the derived data but a possible improvement for future versions of the automatic QC would be to implement some sort of logic check that can prevent these types of mistakes. For the record, the dependency on row 3 is correct while the record on row 6 should be `EVID == 1 & AMT > 0`.

To further speculate on this topic, the current dependency definitions for RATE are redundant. Technically, logically equivalent dependencies could be achieved using only one of the variables EVID or AMT. That is also a possible further improvement for the automatic QC, to warn the user if redundant information is given. On the other hand, for human readers, logically redundant information is sometimes desirable as it may improve readability.

	Variable	Dependencies	PossibleValues	dep	posValue
1	EVID	*NA*	0	TRUE	EVID == 0
2	EVID	*NA*	1	TRUE	EVID == 1
3	AMT	EVID == 0	0	EVID == 0	AMT == 0
4	AMT	EVID == 1	>0	EVID == 1	AMT > 0
5	RATE	EVID == 0 & AMT == 0	0	EVID == 0 & AMT == 0	RATE == 0
6	RATE	EVID == 0 & AMT > 0	-2	EVID == 0 & AMT > 0	RATE == -2

Figure 9. Example of a DDT that would be accepted by the automatic QC module without warnings or errors. However, rows 3 and 6 have contradictory logic in the Dependencies column as row 3 says that AMT must be zero when EVID is 0, while row 6 wants EVID to be 0 and AMT greater than 0.

4.2 R-code vs Human-readable text as input

The current implementation of the automatic QC module forces the input to the 'Dependencies on other variables' field in rock (see Figure 2) to be proper R-code. The main benefit of using R-code as input is that it is logically unambiguous, given that it is typed out correctly. For instance, an R-expression such as `EVID == 0 & TYPE == 0` would, when evaluated by R, always yield the expected result. In addition, it relieves the automatic QC of the responsibility of interpreting any input as it instead expects the user to be responsible for 1) entering a syntactically correct R expression and 2) entering the R expression that was actually intended. For someone who is very comfortable with R, or with programming in general, this may be a fine approach, as it is supposedly robust and unambiguous, even when

read by a human. On the other hand, for users who generally do not work a lot with R, this type of approach may rather seem complicated and unintuitive. The R-expression above, `EVID == 0 & TYPE == 0`, could also, in a typical DDT setting, be interpreted as "an observation record (`EVID == 0`) of some drug A (`TYPE == 0`)", after decoding the variables EVID and TYPE. Recall that EVID means event identifier in NONMEM. That sentence is fully readable for humans, and would probably still be even if the phrasing was different, or if a typo was made. Unfortunately, for a computer, it can be near-impossible to interpret such a sentence unambiguously, especially if it can be phrased in different ways. In short, humans are capable of processing the meaning of texts in a way that is very difficult to implement programmatically. Since correct R-code is unambiguous it should be possible to convert it to human-readable text. The aforementioned sentence `EVID == 0 & TYPE == 0` could quite easily be converted to something along the lines of "observation record and drug A", which is certainly human-readable, but perhaps not a proper sentence. Also, if more complex expressions or combinations of different expressions are involved, simply converting the expressions to words might not qualify as making it human-readable any more.

A possible third approach would be to have the input as a hybrid between R-code and human-readable text. In essence, this would mean that the input is worded as English text but forced to have a certain structure such that it can be interpreted by an algorithm unambiguously. Less strict enforcement of the text structure requires the interpretation algorithm to be more complex and robust, lest you open up for ambiguity. At a certain point however, if the text structure is allowed to be too loose, unambiguous parsing is likely impossible to guarantee.

4.3 Physiologically realistic values

The automatic QC module will, apart from verifying that the data file fulfils DDT specifications, also try to evaluate whether the data is physiologically realistic. This process requires however that the automatic QC can interpret the name of the variable being investigated, interpret the unit which the variable is expressed in and also know what values are physiologically realistic for that variable. Although certain variable names are commonly reused, such as WT indicating body weight and the unit commonly being in kilograms, there is no absolute standard for this and users may freely name their variables and express them in any unit. Further, what is a physiologically realistic value may depend entirely on the setting in which the study was performed. For instance, physiologically realistic body weights are probably entirely different in a paediatric study as compared to a study conducted in adults. Likewise, if a study was conducted on mice, rather than on humans, what constitutes realistic values are also entirely different. Currently, information regarding the study setting is not passed to the automatic QC and therefore, the automatic QC will assume all studies are carried out in human adults. With feedback from independent QC reviewers, future versions of the automatic QC will however attempt to take into consideration the age of patients when inferring whether values are physiologically realistic.

Another possible improvement would be to try and infer which species the study was conducted on depending on the variables present in the dataset. For instance, the variable SMOK, indicating whether the patient is a smoker or not, is likely not used in studies conducted in non-humans. Likewise, a variable ETHN (ethnicity) is sometimes included in datasets for broad classification of patient groups and is also only applicable for humans.

5 Conclusion

This project implemented an automatic QC of DDT:s and data files. The purpose was to facilitate the process of producing data files ready to be pharmacometrically modelled by the software NONMEM, the main analysis software used at Pharmetheus. The process of deriving a NONMEM-ready data file is lengthy, starting with the modellers defining the scope of the project based on a clients request, followed by a data programmer deriving the data file based on a set of source clinical data files. Once the data programmer has finished his/hers task, the derived data file must be scrutinized by a QC team, lest the pharmacometric model risk being inaccurate or even unobtainable owing to potential errors or inconsistencies in the data file. The automatic QC developed will aid the data programmer in discovering certain errors that may be present in the data file as well as helping define a rigorous DDT, and is implemented in Pharmetheus existing graphical user interface for DDT creation. The features of the automatic QC have been described in this paper, how it is supposed to work and what can be expected of it. Further, the current limitations and possible improvements for the future have also been discussed. The definitions for what a pharmacometric data set at Pharmetheus, and the accompanying DDT, should look like is under constant evolution due to feedback from relevant stakeholders. Therefore, the purpose and functionality of the automatic QC is also expected to change over time.

6 Acknowledgement

I would like to thank Kajsa Harling, my supervisor, for her guidance, support and feedback during all stages of this project. I would also like to thank Ola Spjuth, my subject reader, and Anna Odelgard, my opponent, for feedback on the project and report. Special thanks to Rikard Nordgren and Joakim Nyberg, who were not directly involved in this project but nonetheless provided great feedback on the report and helped with acquiring data. In addition, I would like to thank Monique Wouters, for valuable feedback on what could be improved and included in my project.

I would like to thank Jan Andersson and Lena Henriksson for their support and management throughout this project.

Finally, I would like to thank the entirety of Pharmetheus for providing a great and friendly working environment.

References

Atkinson A, Lalonde R. 2017. Introduction of quantitative methods in pharmacology and clinical pharmacology: a histroical overview. Clinical Pharmacology and Therapeutics 82: 3-6.

Beal SL, Sheiner LB, Boeckmann AJ, Bauer RJ. 2014. NONMEM User's Guides. (1989-2014) Icon Development Solutions, Ellicot City, MD, USA.

CDC (Centers for Disease Control and Prevention). 2016. Anthropometric Reference Data for Children and Adults: United States, 2011-2014. US Department of Health and Human Services, Hyattsville, Maryland.

Clinical Data Interchange Standards Consortium. 2018. Study Data Tabulation Model. URL: https://www.cdisc.org/standards/foundational/sdtm. Accessed 2018-12-17.

Clinical Data Interchange Standards Consortium. 2018. Analysis Data Model. URL: https://www.cdisc.org/standards/foundational/adam. Accessed 2018-12-17.

Dabla P. 2010. Renal function in diabetic nephropathy. World Journal of Diabetes 1:48-56.

Fisher D, Shafer S. 2007. Pharmacokinetic and Pharmacodynamic Analysis with NONMEM. NONMEM Workshop, Het Pand, Ghent, Belgium.

Grasela TH, Donn SM. 1985. Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. Developmental Pharmacology and Therapeutics 8: 374-383.

Grolemund G, Wickham H. 2011. Dates and times made easy with lubridate. Journal of Statistical Software 40: 1-25.

Grolemund G, Wickham H. 2017. R for data science: Strings. WWW-dokument 2017: https://r4ds.had.co.nz/strings.html. Accessed 2018-12-17.

Henry L, Wickham H. 2017. rlang: Functions for Base Types and Core R and 'Tidyverse' Features. R package version 0.1.2. URL: https://CRAN.R-project.org/package=rlang.

Karlsson M, Jonsson E, Wiltse C, Wade J. 1998. Assumption testing in population pharmacokinetic models: Illustrated with an analysis of moxonidine data from congestive heart failure patients. Journal of Pharmacokinetics and Biopharmaceutics 26: 207-246.

Lee JY, Garnett CE, Gobburu JV, Bhattaram VA, Brar S, Earp JC. 2011. Impact of pharmacometric analyses on new drug approval and labelling decisions. Clinical Pharmacokinetics 50: 627-635.

Lindom L, Ribbing J, Jonsson EN. 2004. Perl-speaks-NONMEM (PsN)—a Perl module for NONMEM related programming. Computer Methods and Programs in Biomedicine 75: 85-94.

Mohammed M, Jungerwirth S, Asatryan A, Jiang P, Othman A. 2017. Assessment of effect of CYP3A inhibition, CYP induction, OATP1B inhibition, and high-fat meal on pharmacokinetics of the JAK1-inhibitor upadacitinib. British Journal of Clinical Pharmacology 83: 2242-2248.

Mould D, Upton R. 2012. Basic Concepts in population modeling, simulation, and model-based drug development. CPT: Pharmacometrics and System Pharmacology 1: 1-14.

Nordgren R, Freiberga S. 2018. Perl-speaks-NONMEM. URL: https://uupharmacometrics.github.io/PsN/.

R Development Core Team. 2007. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL: http://www.R-project.org.

Standing J. 2017. Understanding and applying pharmacometric modelling and simulation in clinical practice and research. British Journal of Clinical Pharmacology 83: 247-254.

Thomas C, Thomas L. 2009. Renal failure – measuring the glomerular filtration rate. Deutsches Ärtzeblatt 106: 849-854.

Wickham H. 2011. testthat: Get started with testing. The R journal 3: 5-10.

Wickham H, Francois R, Henry L, Müller K. 2017. dplyr: A Grammar of Data Manipulation. R package version 0.7.2. URL: https://CRAN.R-project.org/package=dplyr.

Wickham H, Danenberg P, Eugster M. 2017. roxygen2: In-Line Documentation for R. R package version 6.0.1. URL: https://CRAN.R-project.org/package=roxygen2.

Wickham H. 2017. stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.2.0. URL: https://CRAN.R-project.org/package=stringr.

Zakhari S. 2006. Overview: How is alcohol metabolized by the body. Alcohol Research and Health 29: 245-254.

Appendix 1 Regular Expressions

Regular expressions are string matching techniques that are usually included in most programming languages or environments that allow string matching. Regular expressions can be considered a language for describing patterns in strings. The automatic QC relied heavily on the use of string matching, and therefore regular expressions, to perform interpretation of the DDT. Documentation for regular expressions as used in R can be found in (R Development Core Team 2007). The functions used for string matching are from the R-package stringr (Wickham 2017) and base R (R Development Core Team 2007). Example of how regular expressions are used together with the stringr package:

```
# Pattern that matches any amount of blank -spaces , followed by a ">"
# character, followed by any amount of blank spaces followed by any integer
# preceded by an optional minus sign.
pattern = "^\\ s * >\\ s *-*[0-9]+$"

# the following function detects if the f i r s t argument is recognized by
# the pattern
stringr : : str_detect ("> 0" , pattern )
> TRUE
stringr : : str_detect ("> -5" , pattern )
> TRUE
stringr : : str_detect ("< 10" , pattern )
> FALSE
```

Appendix 2 Regular expressions used for DDT parsing

The regular expression used for parsing of the DDT are the following:

```
pattern = ^\\s*[a-zA-Z]*\\s*(>=|>|<=|<)\\s*(-*)([0 -9]+\\.?[0-9]*|,
variableName,)\\b.*$
matches = >,>=,<,<= followed by any number or known variable name

pattern = ^\\s*([a-zA-z]*)?\\s*(-*[0-9]+\\.?[0-9]*)\\s*(to|TO|To|-)\\s*(-*[0-
9]+\\.?[0-9]*)\\b\\s*$
matches = expression of form : from {any number} to {any number}

pattern = ^.*between\\s*(-*[0-9]+\\.?[0-9]*)\\s*and\\s*(-*[0-9]+\\.?[0-9]*)\\
b.*$
matches = expression of form : between {any number} and {any number}

pattern = ^\\s*={0,2}(-*)([0-9]+\\.?[0-9]*|,variableName,)\\s*$
matches = any number or known variable name

pattern = ^\\ s*(\\w+|\\w+[-_]\\w+|\\w+[-_]\\w+[-_]\\w+|\\w+[-_]\\w+[-_]\
\w+[-_]\\w+)\\s*$
matches = character patterns composed normal letters or numbers separated
by dashes or underscores

pattern = ^\\s*((C|c)haracter)\\s*.*$
matches = the literal string 'character'

pattern = ^\\s*(((N|n)umeric)|((N|n)umerical)|((N| n)umber))\\s*$
matches = the literal strings 'numeric', 'numerical' or 'number'

pattern = &|=|<-|\\|
matches = the characters &, =, <- and | to prevent these from existing in
formulas, should a formula exist in the DDT.

patterns = ([yY]{4}(-)?[mM]{2}(-)?[dD]{2})
([yY]{2}(-)?[mM]{2}(-)?[dD]{2})
([12][0-9]{3}(-)[0][1-9](-)[0123][0-9])
([012789][0-9](-)[0][1-9](-)[123][0-9])
([12][0-9]{3}(-)[1][012](-)[0123][0-9])
([012789][0-9](-)[1][012](-)[123][0-9])
([hH]{2}(:)?[mM]{2}(:)?([sS]{2})?)
([012][0-9](:)[0-5][0-9](:)[0-5][0-9])
([012][0-9](:)[0-5][0-9])
matches = different types of entries to be interpreted as dates or clock
times
```

Appendix 3 Ranges for physiological variables

These are the ranges used for the physiological variables and the recognized variable names at the time of writing the report, which assumes a human adult. Both the names and values are easily configurable.

- Body weight, recognized by variables WGT, WT, BLWT. Accepted range in kilograms: 20-180
- Age, recognized by variables AGE, BLAGE. Accepted range in years: 0-120
- Creatinine clearance, recognized by variable CRCL. Accepted range in ml/min: 15-190
- Serum creatinine, recognized by variable SCR. Accepted range in micromol/l: 45-200
- Height, recognized by variables HT, HGT, BLHT, BLHGT. Accepted range in centimeters: 60-210