WinBUGS Differential Interface – Worked Examples

Dave Lunn
Imperial College School of Medicine, London, UK
d.lunn@imperial.ac.uk

September 1, 2004

Note: Whilst we have endeavoured to ensure that our differential equation solving algorithms are robust, there is no guarantee that a given system of equations can be solved accurately for all possible sets of system parameters. Hence, it is sometimes *essential* to specify informative priors.

Worked Example 1

Our first worked example makes use of a physiologically based pharmacokinetic (PBPK) model for a hepatically cleared drug given orally to some hypothetical animal. The model is for illustrative purposes only, to show the reader how the WinBUGS Differential Interface (WBDiff) may be used – all of the physiological parameter values specified have been fabricated and bear no (intended) relation to any specific drug or animal species. The model is depicted in Figure 1 and can be described mathematically via:

$$\frac{\mathrm{d}C_{PP}}{\mathrm{d}t} = K_{I_{PP}}C_{ART} - K_{O_{PP}}C_{PP} \quad (1)$$

$$\frac{\mathrm{d}C_{RP}}{\mathrm{d}t} = K_{I_{RP}}C_{ART} - K_{O_{RP}}C_{RP} \quad (2)$$

$$\frac{\mathrm{d}C_{GU}}{\mathrm{d}t} = K_{I_{GU}}C_{ART} - K_{O_{GU}}C_{GU} \quad (3)$$

$$\frac{\mathrm{d}C_{LI}}{\mathrm{d}t} = \left(Q_H C_{ART} + K_{O_{GU}}V_{GU}C_{GU} + RA - RM\right)/V_{LI} - K_{O_{LI}}C_{LI} \quad (4)$$

$$\frac{\mathrm{d}C_{LU}}{\mathrm{d}t} = K_{I_{LU}}C_{VEN} - K_{O_{LU}}C_{LU} \quad (5)$$

$$\frac{\mathrm{d}C_{VEN}}{\mathrm{d}t} = \left(K_{O_{PP}}V_{PP}C_{PP} + K_{O_{RP}}V_{RP}C_{RP} + K_{O_{LI}}V_{LI}C_{LI} - K_{I_{LU}}V_{LU}C_{VEN}\right)/V_{VEN} \quad (6)$$

$$\frac{\mathrm{d}C_{ART}}{\mathrm{d}t} = \left(K_{O_{LU}}V_{LU}C_{LU} - Q_{TOT}C_{ART}\right)/V_{ART} \quad (7)$$

Here $C_{\cdot}$ denotes the concentration of drug within the indicated compartment, $K_{I_T} = Q_T/V_T$ and $K_{O_T} = Q_T/V_T K_{P_T}$, where Q_T, V_T and K_{P_T} denote the rate of blood flow, the volume, and the tissue-to-blood partition coefficient associated with tissue/compartment T, respectively. In addition, Q_H represents the liver's *arterial* blood supply ($Q_H = Q_{LI} - Q_{GU}$; see Fig. 1) and Q_{TOT} is the total cardiac output, which is equal to Q_{LU} as all blood must flow through the lungs to be oxygenated. (Note that for mass balance we must also have that $Q_{TOT} = Q_{LU} = Q_{PP} + Q_{RP} + Q_{GU} + Q_H$.) Also, for this particular example, the rate of absorption is given by $RA = k_a \times F \times Dose \times \exp(-k_a t)$ (first-order absorption) and the rate of metabolism, RM, is given by $V_{max} \times C_{LI}/(K_m + C_{LI})$ (Michaelis-Menten metabolism), where k_a, F, V_{max} and K_m are the first-order absorption rate constant, the fraction of the administered dose that enters the liver, the maximum rate of metabolism, and the concentration of drug in the liver at which exactly half the maximum rate of metabolism occurs, respectively.

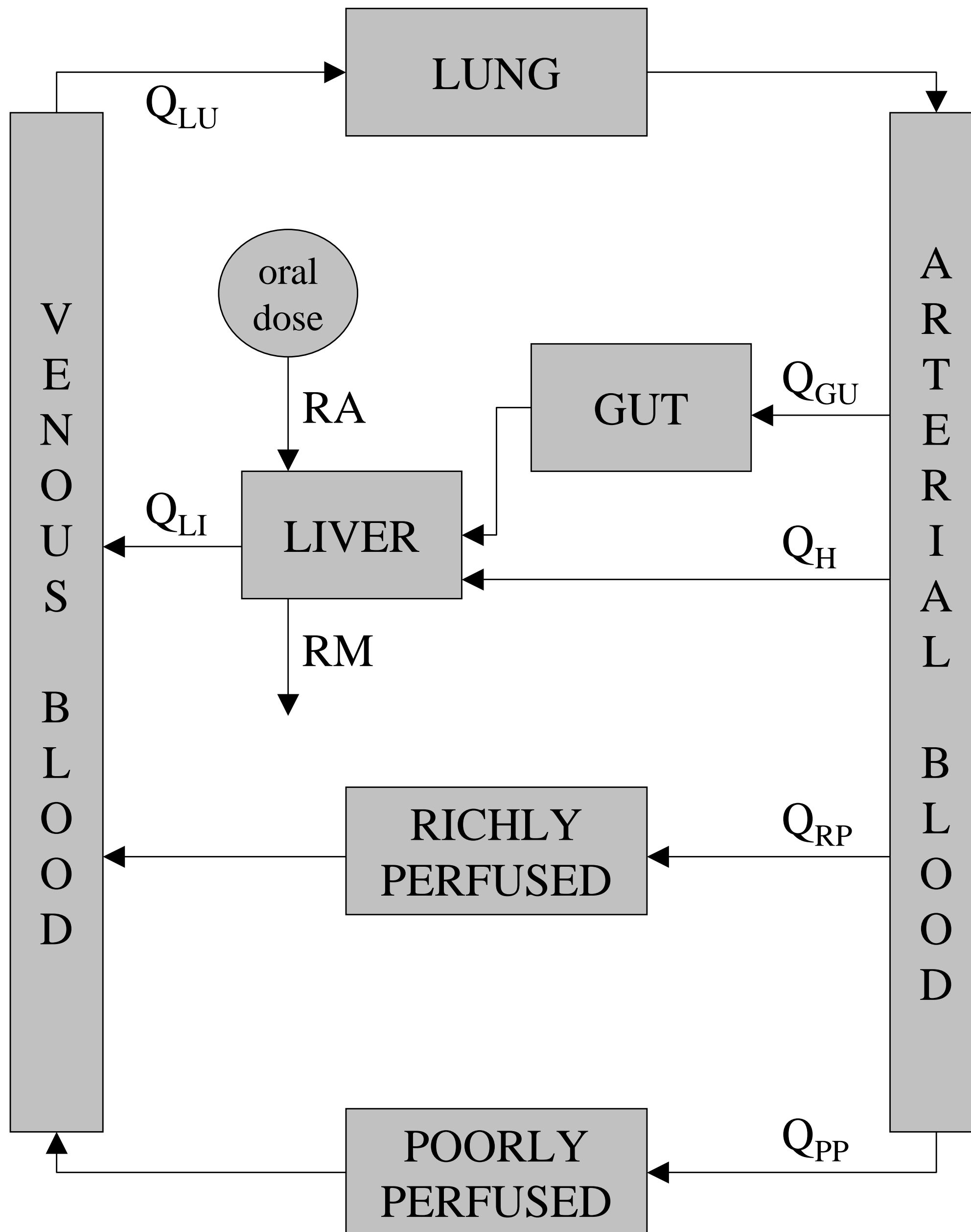

Figure 1: Physiologically based pharmacokinetic model for a hypothetical, orally administered, hepatically cleared drug. Note that orally administered drug is generally absorbed across the gut wall into the hepatic portal vein. In order to separate any drug that is being absorbed thus from that which is *recirculating* into the hepatic portal vein, via the tissues of the gut, we tend to assume that as far as the model is concerned oral absorption processes input drug directly into the “LIVER” compartment, as shown, rather than the “GUT” compartment.

This first worked example comes in two parts: in 'part 1' we use the above model to simulate a small set of data; in 'part 2' we use the same model to perform Bayesian inference on those data.

Worked Example 1, Part 1: Simulation

Start your copy of WinBUGS 1.4 and open the file `WBDiff/Examples/WBDiff_example_part1.odc` from within it. The following notes describe/explain various parts of the model code – the line numbers to which they relate are given in the right-hand margin of the `WBDiff_example_part1.odc` file. The file contains code for both parts of the worked example, colour-coded as follows: code that is only relevant to part 1 of the worked example is shown in blue; code that is only relevant to part 2 of the worked example is shown in red; and code that is relevant to both parts is shown in black. (Note that all of the red code is commented out – using '#' at the beginning of each line – for this first part of the example.)

Notes

Line 2: `n.grid` is the number of time-points at which the solution to the differential equations is required; `dim` is the number of differential equations to be solved. `ode(.)` is the BUGS-language syntax for the new differential equation solver. Its five arguments are described as follows: (i) `init[.]` contains the initial conditions for each compartment, which pertain to the time-origin specified via `ode`'s fourth argument, `origin`; (ii) `grid[.]` is the grid of time-points at which the solution is required; (iii) `D(C[.], t)` is the system of differential equations to be solved; (iv) `origin` denotes the time-origin of the system, i.e. to when the initial conditions apply ($t = 0$ in this example); and (v) `tol` is the required level of accuracy (1×10^{-3} here).

Lines 3–10: Specification of the system of ordinary differential equations (ODEs) to be solved. Hopefully the notation is self-explanatory. The use of compartment names, rather than numbers, to index the various vectors involved helps make the system more 'readable' and thus probably reduces the likelihood of coding errors. These compartment names are mapped to numbers in the data block beneath the model code (e.g. `PP = 1`, `RP = 2`, etc.) Note that the rate of oral absorption (`RA`) and the rate of metabolism (`RM`) are defined outside this equation block (on lines 16–18) – this enhances clarity and also facilitates modification of the code to suit different circumstances.

Line 21: `five.comp.model(.)` is a new BUGS-language component that provides the link to a 'hard-wired' (i.e. compiled Pascal) version of the five compartment PBPK model described above – see "New BUGS-language component" below for implementation details. This type of component provides a much faster means of evaluating the model but is obviously much less straightforward for the user to implement themselves. The new component's five arguments are the same as for the more general `ode(.)` component described above, except that a 22-dimensional parameter vector, `theta[.]`, replaces the set of system equations denoted previously by `D(C[.], t)`. The elements of `theta[.]` are defined on lines 22–27.

Line 30: We want our data set to comprise simulated concentrations of drug in venous blood and so we specify `solution[i,VEN]` as the mean for each `data[i]` (`i` $= 1, ...,$ `n.grid`).

Lines 32 & 33: For simulating the data we specify a fixed value of 10000 (`tau.simulate`) for the `tau` parameter whereas for Bayesian inference (part 2) we specify a (minimally informative) gamma prior.

Lines 34–37: A log-normal prior with a precision, on the log-scale, of `log.param.prec` $=$ `13.81` is specified for each K_{P_T}, $T = 1, ..., 6$. This level of precision corresponds to up to two-fold variation (from the median) across the central 99 per cent of the distribution – thus the prior probability that a given partition coefficient is greater than twice or less than half the value of the prior median is 0.01. The prior median is specified via prior point estimates for each K_{P_T} contained in `data.KP[.]` – see the block of data beneath the model code. Part 1 of our worked example is merely a Monte Carlo simulation exercise, i.e. the sampling distribution for each stochastic variable is simply the distribution specified for that variable in the model. In contrast, in part 2, where we have

'observed' some data and wish to use them to perform Bayesian inference (via *Markov chain* Monte Carlo simulation – MCMC), the sampling distributions of unobserved stochastic variables are modified by the observed data. Thus in part 1 our prior merely acknowledges a level of uncertainty to be directly reflected in the simulated concentrations (`solution[.,.]`) whereas in part 2 it represents *a priori* uncertainty to be converted into *posterior* uncertainty via the extraction of relevant information from the observed data.

Lines 38–39: The prior specified for `log.ka` is exactly analogous to that specified for each element of `log.KP[.]`.

Running the simulation

In order to reproduce the selected results shown at the foot of the `WBDiff_example_part1.odc` file, you should monitor the `log.KP`, `log.ka` and `solution` variables and run the simulation for 1000 iterations – this took around three minutes on a 2 GHz laptop machine. The plot is of simulated venous blood concentrations versus time and was generated by entering `solution[,6]` and `grid` in the `node` and `axis` fields, respectively, of the `Comparison Tool` dialogue box (select `Compare...` from the `Inference` menu), and by then pressing the `model fit` command button. The data set to be used in part 2 of our worked example is given by the current value of the `data` vector and can be obtained by selecting `Save State` from the `Model` menu – note that the current values of `log.KP[.]` and `log.ka`, i.e. those used to generate the data, are also given.

Worked Example 1, Part 2: Inference

The WinBUGS code required in order to run part 2 of this worked example, i.e. Bayesian inference from the data generated in part 1, can be obtained by simply commenting out all of the blue code in `WBDiff_example_part1.odc` and by also removing the comment markers from the beginning of each line of red code; the data block beneath the model code also requires some minor alterations. For your convenience the file `WBDiff/Examples/WBDiff_example_part2.odc` incorporates all of the required modifications. In order to generate the results shown at the foot of this file, we first set the length of the WinBUGS Metropolis algorithm's adaptive phase equal to 2000 iterations (see "The Options Menu – Update options..." in the WinBUGS manual for details) and then we ran WinBUGS for 4000 iterations to obtain 2000 posterior samples (iterations 2001–4000). This took under 10 minutes on a 2 GHz laptop machine (with the BUGS-language specification of the PBPK model it would have taken around 90 minutes). The plot shown is of 'model predicted' together with 'observed' venous blood concentrations against time – this was generated by entering `solution[,6]`, `data` and `grid` in the `node`, `other` and `axis` fields, respectively, of the `Comparison Tool` dialogue box, and by then pressing the `model fit` command button. In what follows we discuss the implementation of the new `five.comp.model(.)` component (see line 21 of `WBDiff_example_part2.odc`), with an emphasis on how to use the underlying Pascal code as a template for creating further new components.

New BUGS-language component

The new component can be found in the file `WBDiff/Mod/FiveCompModel.odc`. This is a *Component Pascal module* that under normal circumstances, i.e. if you had written it yourself, would need to be compiled and then 'linked' into the core WinBUGS software before it could be used from within the software – I have already done this for this example, however, so that the statistical model in `WBDiff_example_part2.odc` can be analysed immediately. Before discussing the new module in detail we provide instructions on how to set up your system so that Component Pascal code can be compiled.

1. Download *BlackBox Component Builder* from the following web-page:
   `http://www.oberon.ch/blackbox.html`

2. Unzip the downloaded file, if necessary. Install 'BlackBox' by double-clicking on the `Setup.exe` icon and following the instructions. The software should be installed into the new directory `Program Files/BlackBox`.

3. Open `My Computer` (or its equivalent) and navigate to the `Program Files/WinBUGS14` directory; then press `Ctrl+A` (or select `Select All` from the `Edit` menu) to select all files and sub-directories within the `WinBUGS14` directory. Now press `Ctrl+C` (or select `Copy` from the `Edit` menu) to copy those files and sub-directories.

4. Continue using `My Computer` to navigate to the `Program Files/BlackBox` directory and then press `Ctrl+V` (or select `Paste` from the `Edit` menu) to paste the copied files and sub-directories to this location. Select "Yes to All" if prompted about replacing existing files.

5. Now your copy of BlackBox should include the *full* functionality of WinBUGS 1.4 within it, and so the `BlackBox.exe` icon on the desk-top or that in the `Program Files/BlackBox` directory can be used either to run WinBUGS in the normal way or to conduct WinBUGS development work (or even more general Component Pascal programming).

Now we discuss the new BUGS-language component coded in `WBDiff/Mod/FiveCompModel.odc` – don't forget to open the file via BlackBox if you wish to modify and/or recompile it. Note that to reduce the risk of errors creeping into the system we recommend that all other new components are also stored in the `WBDiff/Mod` directory. The `FiveCompModel` component can be used as a template for such new components – only those areas of the code currently marked in blue should be modified. The following notes pertain to areas of the code labelled with the relevant numbers in brackets: `(*.*)`

- `(*1*)` First note that in Component Pascal comments should be enclosed within `(*` and `*)`, e.g. `(* this is a comment *)`. The first line of a Component Pascal module should always read `MODULE`, followed by the module's name, in this case `WBDiffFiveCompModel`, followed by a semi-colon. The last line of the module should read `END`, followed by the module's name, followed by a *full stop*. All new module names for new BUGS-language components of this type should begin with `WBDiff`; the corresponding file names should be identical but with the `WBDiff` prefix removed (they must also begin with at least one capital letter); all new files of this type should be saved in the `WBDiff/Mod` directory.

- `(*2*)` Various other modules can be 'imported' into each new module, which means that procedures and/or data structures defined in those modules can be used/exploited from within the new module. The `Math` module is an integral part of the BlackBox software since it defines many fundamental mathematical functions, which are called from within other modules via the syntax `Math.` followed by the relevant procedure name, e.g. `Math.Ln(.)` for natural logarithms, `Math.Exp(.)` for exponentials – see lines `(*19*)`, `(*22*)` and `(*23*)`. Documentation regarding the `Math` module can be accessed by highlighting the word `Math` and selecting `Documentation` from the *second* `Info` menu in BlackBox.

- `(*3*)` `nEq` should be set equal to the number of equations to be solved, in this case 7 – one for each compartment plus one for venous blood and one for arterial blood.

- `(*4*)`–`(*7*)` Here we define meaningful names with which to index the system's 22-dimensional parameter vector (note that array indices in Component Pascal begin at 0 rather than 1). The ordering of the parameters is arbitrary but, obviously, the same ordering should be used within the statistical model, i.e. on lines 22–27 of `WBDiff_example_part2.odc` (see earlier). We use the 'underscore' character ('_') rather than a full stop ('.') to separate the various parts of each name since '.' has a special meaning in Component Pascal.

- `(*8*)` As in part 1 of our worked example (see lines 3–10 of `WBDiff_example_part1.odc`), we prefer to index compartments via names rather than numbers and so here we provide a mapping between the desired names and the indices to which they relate – obviously the same ordering should be used within the statistical model in `WBDiff_example_part2.odc`.

(*9*)–(*10*) The `Derivatives` procedure defines the system of ordinary differential equations to be solved. It has four 'input' arguments, `theta[.]`, `C[.]`, `n` and `t`, and one 'output' argument, `dCdt[.]`. These are described as follows: `theta[.]` is the 22-dimensional parameter vector (in general, any number of parameters is permissible, so long as the vector has the same length here as it does in the statistical model); `C[.]` represents the set of compartmental concentrations; `n` is the number of equations to be solved (you can probably ignore this); and `t` is the variable with respect to which derivatives are calculated – 'time' in this case. Any of these input parameters can be used at any point within the `Derivatives` procedure proper (lines `(*16*)–(*34*)`) in order to help define the system equations, which are passed out of the `Derivatives` procedure via the `dCdt[.]` variable. The output vector `dCdt[.]` is defined by setting each of its elements equal to one of the system equations (as shown on lines `(*26*)–(*34*)`).

(*11*)–(*14) Note that any number of 'local' variables can be declared and used to aid in specifying the system equations, so long as their names do not clash with other variable/procedure names – the compiler (`Ctrl+K`) will normally inform the programmer of any errors.

(*15*)–(*35*) This part of the code is intended to be reasonably self-explanatory and hopefully demonstrates sufficient use of the Component Pascal syntax that the reader is able to write their own module from this template. **Please note that (almost) every Component Pascal statement ends with a semi-colon.**

Using "WBDiffFiveCompModel" as a template

The following steps should be followed closely when defining a new BUGS-language component via the `FiveCompModel` template:

1. Choose a name for the new component, `NewComponent`, say (the new name *must* begin with a capital letter). Start your copy of BlackBox and open the `WBDiff/Mod/FiveCompModel.odc` template from within it; then save the template under the new name, e.g. `WBDiff/Mod/NewComponent.odc` – **be careful not to overwrite an existing module!** Now modify the module name both at the top and at the bottom of the new file – change these from `WBDiffFiveCompModel` to `WBDiff` followed by the new component's name, e.g. `WBDiffNewComponent`. Save and compile the new component by pressing `Ctrl+S` (save) followed by `Ctrl+K` (compile) – there should be no compilation errors at this stage since only the module name has been changed.

2. Now modify the code in the new module according to your desired model. You can save the new module at any time by pressing `Ctrl+S` (or by selecting `Save` from the `File` menu). You can also attempt to compile the code at any time by pressing `Ctrl+K` (or by selecting `Compile` from the `Dev` menu). If there are any compilation errors when you attempt to compile your code, each one will be marked in the code by a grey box with a white cross running through it. An error message pertaining to the first error will be displayed on the status bar (which lies across the bottom of the BlackBox 'program window') and the cursor should automatically position itself next to the corresponding grey box. We advise that you deal with any compilation errors in order, but if for some reason this makes things awkward (or is not possible) then error messages for specific compilation errors can be obtained by clicking on the appropriate grey boxes – a single click shows the error message on the status bar whereas double-clicking reveals it within the code, in place of the grey box (double-click again to revert back to the grey box).

3. Once the new module has been successfully compiled (and saved) then it can be 'linked' into the WinBUGS software by modifying the file `WBDiff/Rsrc/Grammar.odc`. The first line of this file contains the required entry for the `five.comp.model(.)` component:

   ```
   v <- "five.comp.model"(v, v, v, s, s)      "MathRungeKutta45.Install; WBDiffFiveCompModel.Install"
   ```

   Make a copy of this line immediately beneath it, and replace `five.comp.model` on the *new* line with the desired BUGS-language name for your new model, e.g. `new.component` (this is the name with which you wish to refer to the new model during future WinBUGS sessions). Now specify the name of the module where the new model's 'installation' procedure can be found: replace

`WBDiffFiveCompModel` with your new module's name, e.g. `WBDiffNewComponent`. You should end up with something like

```
v <- "five.comp.model"(v, v, v, s, s)     "MathRungeKutta45.Install; WBDiffFiveCompModel.Install"
v <- "new.component"(v, v, v, s, s)       "MathRungeKutta45.Install; WBDiffNewComponent.Install"
```

at the top of the `WBDiff/Rsrc/Grammar.odc` file. Now save the new `WBDiff/Rsrc/Grammar.odc` file – the new component will be available from the next time that BlackBox is started, so don't forget to shut down the software before trying to use your new model. **Good luck!**

Worked Example 2

The physiologically based pharmacokinetic model used in the previous example(s) is such that all of the first derivatives are *smooth* functions of time[1] (i.e. the second derivatives, with respect to time, are all continuous). WBDiff is capable of handling situations where this is not the case but it requires additional information regarding the times at which discontinuities occur. It needs this information so that it can break up the problem into "blocks" of time where, throughout each block, all second derivatives are continuous. Because specifying such models is somewhat more complicated than with their smooth counterparts, WBDiff provides an additional BUGS-language component (`ode.block(.)`) and an additional template module (`WBDiffWorkedExample2`) dedicated to this task. This worked example demonstrates their usage; we begin by describing the model.

Population PK Model

Our model for this example is designed to illustrate several features of 'advanced' WBDiff use, such as the specification of population models, handling discontinuities in time, and allowing compartments to empty into each other, for example. In pharmacokinetics, a population model is required when, in order to draw inferences about the target population[2], a study is conducted whereby a number of healthy volunteers or patients each receive one or more doses of the drug under investigation and concentration measurements are taken from each one. Typically we have a (parametric) structural model that describes the shape of the concentration-time profile and we model variability throughout the study population by allowing the parameters of that structural model to differ between individuals. The structural model for this example is shown in Figure 2. The human body is represented by a single compartment ("Compartment 1") and as such is assumed to be homogeneous – the blood and all organs/tissues contain the same concentration of drug, C, which is given by the total amount of drug in the body at time t, $A_1 = A_1(t)$, divided by an *apparent* volume of distribution, V. Drug is eliminated (or *cleared*) from the body via a 'flow-rate' of CL (*clearance*) – the rate of elimination, i.e. the rate at which drug leaves the body, is given by $\frac{dA_{el}}{dt} = CL \times C = CL \times A_1/V$. Compartments 2 and 3 contain doses of drug to be administered, to Compartment 1, at times $t = t_{b2}$ and $t = t_{zo}$, respectively. These have been incorporated purely for illustrative purposes – they are actually unnecessary as we could easily specify the model using Compartment 1 alone (with an appropriate sequence of initial conditions for each block of time).

The full sequence of events that generates our model is described as follows. At time $t = 0$, the volunteer/patient receives an intravenous bolus dose[3] of size D. At time $t = t_{b2} > 0$ a second bolus, of the same size, is administered. This is the process represented by the dashed arrow from Compartments 2 to 1 in Figure 2 – dashed arrows represent processes whereby one compartment empties into another instantaneously at the specified time ($t = t_{b2}$ in this case) whereas normal arrows denote *rates* of movement of drug. At time $t = t_{zo} > t_{b2}$ a zero-order process is initiated between Compartments 3

[1] Note that, in general, derivatives may be defined with respect to any suitable quantity, e.g. temperature, concentration of substrate. In situations where a variable other than time is appropriate, then the name of that variable can be substituted for "time" in the text.

[2] The target population comprises those individuals who will receive the drug therapeutically.

[3] Intravenous boluses are such that the whole dose is administered, as quickly as possible, directly into venous blood – the time taken to deliver the dose is generally modelled as zero and the rate of input is effectively infinite.

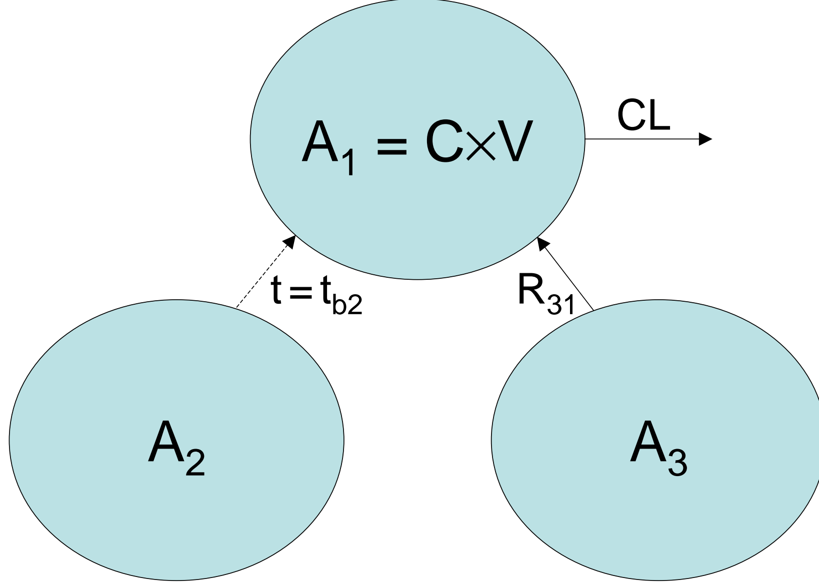

Figure 2: Structural three compartment model used in Worked Example 2 – see text for details.

and 1 whereby the dose initially in Compartment 3 (again, D) is transferred to Compartment 1 at a constant rate given by D/TI, where TI denotes the duration of the zero-order input process. Clearly this process must terminate at $t = t_{zo} + TI$. All of this means that the somewhat abstract quantity R_{31} in Figure 2, which simply represents the rate of movement of drug from Compartments 3 to 1, is a *piecewise smooth* function of time. Explicitly, it is given by

$$R_{31} = \begin{cases} 0 & 0 \le t < t_{b2} \\ 0 & t_{b2} \le t < t_{zo} \\ D/TI & t_{zo} \le t < t_{zo} + TI \\ 0 & t \ge t_{zo} + TI \end{cases} \tag{8}$$

More generally, we can write $R_{31} = pw(v, o, t)$, where v and o are vectors containing each smooth function and the times at which they begin, respectively; in this case, $v = (0, 0, D/TI, 0)$, $o = (0, t_{b2}, t_{zo}, t_{zo} + TI)$ and $pw = v_b$, where

$$b = \sum_i i \times I(t \in [o_i, o_{i+1})) \qquad (o_5 = \infty), \tag{9}$$

that is, b simply specifies which 'block' of time, defined by a pair of 'change-points', contains the current value of t. WBDiff provides a new BUGS-language component called `piecewise(.)` that can be used for specifying arbitrary piecewise-smooth functions within WBDiff models. Only the vector v is required as an input parameter, however. This is because the 'origins' in o are passed into the associated `ode.block(.)` component instead, so that it knows where to break the problem up into 'smooth' sub-problems (t is defined/controlled by the ODE solving algorithm). For this reason, the o vector passed into `ode.block(.)` must comprise the union of all 'change-points' in the model and each instance of `piecewise(.)` must be defined in terms of a smooth function vector (v) of the same length. This is why R_{31} above incorporates components for both $[0, t_{b2})$ and $[t_{b2}, t_{zo})$ even though it has the same value throughout both intervals: `ode.block(.)` needs to know that a discrete event occurs at $t = t_{b2}$ and so all of its piecewise parents (just R_{31} in this case) must be 'split' accordingly.

The system equations are as follows:

$$\frac{\mathrm{d}A_1}{\mathrm{d}t} = R_{31}(t) - CL \times A_1/V \tag{10}$$

$$\frac{\mathrm{d}A_2}{\mathrm{d}t} = 0 \quad (11)$$

$$\frac{\mathrm{d}A_3}{\mathrm{d}t} = -R_{31}(t) \quad (12)$$

(Note that neither bolus dose is represented in the system equations. This is because bolus doses are instantaneous and are thus best described via the initial conditions, as we discuss below.) The structural model is completed by the specification of a sequence of initial conditions for each compartment. First in the sequence are the initial conditions proper, which pertain to the time origin given by o_1. Note that the easiest way in which to model the first intravenous bolus dose is to simply specify $A_1(o_1) = A_1(0) = D$ in the initial conditions. At each subsequent origin (e.g. o_2, o_3, o_4) we may, generally speaking, wish to apply an instantaneous adjustment to the system to account for certain types of discrete event that may have occurred, such as the administration of a bolus dose (which cannot be represented by a finite rate). In WBDiff we specify such adjustments by declaring an amount (of whatever quantity the differential equations are to be solved for) to be added to each compartment at the appropriate time. If no adjustment is specified (or if a value of zero is declared) then the relevant part of the system is left unchanged. Thus for the example above, we specify the following matrix of 'initial conditions', where rows correspond to origins (o_1, o_2, o_3, o_4) and the columns represent compartments.

$$\begin{pmatrix} D & D & D \\ +A_2 & -A_2 & \mathrm{NA} \\ \mathrm{NA} & \mathrm{NA} & \mathrm{NA} \\ \mathrm{NA} & \mathrm{NA} & \mathrm{NA} \end{pmatrix} \quad (13)$$

where the "NA"s denote missing values. The $\pm A_2$ on row 2 represents $\pm$ the solution to the second differential equation at the relevant time, i.e. $t = t_{b2}$ (the second 'origin'). Thus the amount of drug in Compartment 2 instantaneously changes from $A_2(t_{b2-})$ to zero at $t = t_{b2}$ (because its value is reduced by $A_2(t_{b2-})$). In contrast, the amount of drug in Compartment 1 changes from $A_1(t_{b2-})$ to $A_1(t_{b2-}) + A_2(t_{b2-})$. In other words, Compartment 2 instantaneously empties into Compartment 1 at time $t = t_{b2}$. Since Compartment 2 initially contains an amount D of drug and $\frac{dA_2}{dt} \equiv 0$, this is equivalent to an intravenous bolus dose being administered, to Compartment 1, at $t = t_{b2}$.[4]

The *statistical* model is defined as follows. First note that the measurable quantity here is *concentration* of drug in Compartment 1, i.e. $C = A_1/V$. This is a function of the dose D, time t, and additional parameters, namely CL, V and TI, that we collectively denote by θ: $C = C(\theta, t, D)$. For this example we will allow both θ and D to vary between individuals, although the doses are assumed known, as they would be in practice (normally). Suppose we have n concentration measurements, indexed by j, from each of K individuals, indexed by i. We denote these by y_{ij}, $i = 1, ..., K$, $j = 1, ..., n$. Often in pharmacokinetics, the size of the error associated with each concentration measurement is proportional to the underlying true concentration and so we tend to model the natural logarithm of the data as a function of $\log C$. At the first stage of the statistical model we assume

$$\log y_{ij} = \log C(\theta_i, t_{ij}, D_i) + \epsilon_{ij}, \qquad i = 1, ..., K, \quad j = 1, ..., n, \quad (14)$$

where t_{ij} denotes the time at which y_{ij} was collected and the ϵ_{ij} terms are independent and identically distributed normal random variables with mean zero and variance τ^{-1} (i.e. precision = τ). At the second stage of the model we have

$$\theta_i \sim \mathrm{MVN}_3(\mu, \Omega), \qquad i = 1, ..., K, \quad (15)$$

where μ and Ω are the population mean and the variance-covariance of pharmacokinetic parameters, respectively. At the *final* stage of our population model, appropriate multivariate normal, Wishart and gamma priors are specified for μ, Ω^{-1} and τ, respectively. WinBUGS code for this model is provided in the file `WBDiff/Examples/WBDiff_example2.odc`, which is discussed in detail below.

[4] Alternatively, in the absence of Compartment 2, we could have modelled the second bolus dose simply by specifying a D on row 2 of column 1, to indicate that an amount D should be added to Compartment 1 at $t = t_{b2}$, i.e. $A_1 \to A_1(t_{b2-}) + D$.

Worked Example 2, WinBUGS code

The following notes describe/explain various parts of the model code for this worked example – the line numbers to which they relate are given in the right-hand margin of the `WBDiff_example2.odc` file.

Lines 2, 6 & 7: `n.grid` and `n.ind` are the WinBUGS variables used to store the values of n and K, respectively. For this example, `n.grid` $= n = 14$ and `n.ind` $= K = 10$.

Lines 2–5: Here we specify two variables, namely `s1[1:n.grid]` and `s2[1:n.grid]` that are required for defining the analytic solution for C, which is readily available for this example – see the notes for lines 11–15 below. Both variables represent 'shifted' time: `s1[j]` (`j` $= 1, ...,$ `n.grid`) is given by `grid[j]` minus `second.bolus.time`, where `second.bolus.time` $= t_{b2}$ and `grid[j]` represents t_{ij} (note that all individuals share the same grid of time points, and so, for each j, $t_{ij} = t_j$ for all i); `s2[j]`, on the other hand, is defined as `grid[j]` minus `zo.start.time` where the latter variable represents t_{zo}.

Lines 8–10: `log.data[i,j]` and `log.model[i,j]` represent the natural logarithms of y_{ij} and $C(\theta_i, t_{ij}, D_i)$, respectively. In this worked example, we specify the value(s) of C in three different ways: (i) the analytic solution; (ii) via BUGS-language representation of the underlying ordinary differential equations; and (iii) via a hard-wired Component Pascal representation of the underlying equations. We can use any one of these in order to model the data, although the time taken to run the analysis will, of course, depend on that choice. Here we have chosen to specify our population model in terms of the BUGS-language ODE representation, which is represented by the variable `c.language[.]` – see below.

Lines 11–15: Here we specify the three different ways of defining $C(\theta_i, t_{ij}, D_i)$. On lines 11–13 we specify the analytic solution, which we denote by `c.analytic[.]`. This is given by the sum of three 'time-shifted', 'closed-form', 'single-dose' pharmacokinetic models, one for each dose received by the volunteers/patients. The functions `wbdiff.pkIVbol1(.)` and `wbdiff.pkZO1(.)` have been 'borrowed' from the *PKBugs* software [1]. Although detailed information about them can be found in the PKBugs documentation, this is unnecessary for our worked example as we only use the analytic solution to check that the ODE-solving capabilities of WBDiff are working correctly. On lines 14 and 15, we define `c.language[i,j]` and `c.hard.wired[i,j]` as being given by `a.language[i,j,1]` and `a.hard.wired[i,j,1]`, respectively, divided by the relevant individual's volume parameter, `V[i]`. `a.language[.]` and `a.hard.wired[.]` hold the numerical solutions to the differential equations specified via the BUGS language and via compiled Pascal code, respectively; `a.language[.,.,1]` and `a.hard.wired[.,.,1]` represent the amounts of drug in Compartment 1 derived from each approach.

Lines 16 & 17: The `e[.]` variable is defined simply to check that the ODEs are being solved correctly. Predicted concentrations based on both the BUGS-language specification and on the 'hard-wired' specification of the differential equations are subtracted from the analytic solution to form error measurements.

Lines 19 & 20: The first `p` (= 3) elements of each row of the `theta[.]` matrix represent θ_i whereas the fourth element contains D_i for convenience. This is because we need to pass D_i to our hard-wired component along with the stochastic parameters in θ_i so that it can be defined properly.

Lines 24–27: Use of the `ode.block(.)` BUGS-language component and of analogous (i.e. 'blocked') hard-wired components is virtually the same as before (i.e. for `ode(.)` and related new components). The only differences are those alluded to in the mathematical description of the model given above. First, for each set of equations (i.e. for each individual in this example) the single time origin specified previously is replaced by a *vector* of 'origins' (denoted `origins[.]` here) that define the blocks of time throughout which the differential equations are smooth. Second, for the BUGS-language specification, we replace the vector of initial conditions required previously, for each set of equations, with a matrix (denoted `inits[.]` here) whose rows correspond to origins and whose columns represent compartments/equations. Note that except for the first row of this matrix, "initial conditions" here means 'potential additive adjustments to the system' as described above.

Note in particular that only those rows of `inits[i,.,.]` that contain non-zero adjustments need be specified[5] – for this example, only `inits[i,1,.]` and `inits[i,2,.]` have been defined for each individual (see lines 36–40 and line 25) as no other adjustments are necessary.

For hard-wired systems of 'blocked' differential equations, we only pass the initial conditions proper into each specialized component, as before – hence the first argument to `ode.example2(.)` for individual `i` is `inits[i,1,1:dim]`, i.e. just the first row of the `inits[i,.,.]` matrix. This is because we make all necessary adjustments to the system *internally*, i.e. within the new module. Any additional parameters that are required in order to define such adjustments should be passed into the hard-wired component via the `theta[i,.]` vector – no such parameters are required for this worked example, however. We discuss hard-wired components in more detail below.

Hopefully, the remainder of the WinBUGS model code is self-explanatory. Following the code are three blocks of input information, which are labelled "Data (inference)", "Initial values" and "Data (simulation)". Clearly the "Initial values" section defines initial values for the MCMC analysis. The reason that there are two blocks of data, however, is that the data to be used to run the example, "Data (inference)", were originally simulated using the "Data (simulation)" block and the latter is given, towards the foot of the file, for completeness. In particular, individual θ_i vectors were generated from a trivariate normal distribution with mean and inverse-covariance given by the specified values of `mu[.]` and `omega.inv[.]`, respectively. Then, log-concentration measurements (`log.data[.]`) were generated by simulating from

$$\mathrm{N}(\log C(\theta_i, t_{ij}, D_i), \tau^{-1}), \qquad \tau = 100. \tag{16}$$

In practice, this is achieved by simply: (i) specifying the same model as for the analysis (except that lines 46–49 are removed); (ii) loading the data contained in "Data (simulation)" and compiling; (iii) clicking on the "`gen inits`" button in the `Specification Tool` dialogue box; (iv) running the simulation for *any* number of iterations > 0 (we used 100); and, finally, (v) selecting "`Save State`" from the `Model` menu. Some summary statistics from analysis of these simulated data are given at the foot of the `WBDiff_example2.odc` file.

Worked Example 2, "WBDiffWorkedExample2" template

Component Pascal code for the system of differential equations used in this worked example can be found in the file `WBDiff/Mod/WorkedExample2.odc`. This module is linked into the WinBUGS software via the new BUGS-language component `ode.example2(.)`. The linking procedure for new modules based on this template is virtually identical to that for new components based on the `WBDiffFiveCompModel` template (see "Using WBDiffFiveCompModel as a template"). However, there is one very minor, albeit very important, difference: the argument list for a 'blocked' system of ODEs must be "`v, v, v, v, s`" rather than "`v, v, v, s, s`", to reflect the fact that a single time origin has been replaced by a vector of origins. You can ensure that the correct argument list is specified in `WBDiff/Rsrc/Grammar.odc`, when linking a new component into the software, by always copying and modifying the entry pertaining to the relevant original template, i.e. "`v <- "five.comp.model"(...`" for smooth systems and "`v <- "ode.example2"(...`" for *piecewise smooth* systems.

We close this worked example with a few notes on the Component Pascal code contained within `WBDiffWorkedExample2`. Each note pertains to lines of code labelled with the relevant number(s) in brackets: `(*.*)` in the left-hand margin of `WBDiffWorkedExample2`.

Lines 2 & 19–21: It is a convention that we have adopted to define hard-wired systems of differential equations in terms of an abstract vector quantity `C[.]`. Be careful not to confuse this with 'concentration' – the differential equations in `WBDiffWorkedExample2` are defined in terms of *amounts* of drug regardless of what might otherwise be implied by the notation.

[5] The first row of each initial conditions matrix must be fully specified as it contains the initial conditions proper.

Lines 5, 8 & 13–18: As discussed previously, WBDiff splits the time-grid up into m, say, (user-specified) blocks of time, throughout which it can assume all of the ODEs to be smooth. The `e.Block()` procedure automatically returns an integer value informing us of which block of time is currently of interest to the ODE solver; we use the integer variable '`block`' in order to store this value, an integer between zero and $m-1$ (inclusive). This can be used in order to define any piecewise functions that may be required to fully specify the equations. On lines 13–18 we make use of a Component Pascal "CASE" statement to define the value of `R31`: for each possible value of `block` $(0, 1, 2, 3)$, a corresponding value for `R31` is specified. (More generally, arbitrary statement sequences can be specified for each possible value of the 'CASE variable'.)

Lines 23–36: The "`Adjust`" procedure allows us to specify additive adjustments to the system of differential equations to be applied at the beginning of specific blocks of time – the procedure is called automatically by WBDiff as soon as the ODE solver begins work on a new 'block'. Again we make use of an integer variable named `block`, and of the `e.Block()` procedure, to store the index of the current block of time. This is then used to form a CASE statement whereby any necessary adjustments can be associated with the correct change-point/origin. Note that within the `Adjust` procedure the possible values of `block` are $1, 2, ..., m-1$; that is, adjustments cannot be made at the beginning of 'block 0' because that is the time origin, when the initial conditions proper apply. Note also that 'empty statement sequences' can be specified if no adjustment is required, as is the case at the beginning of the blocks indexed by `block = 2` ($t = t_{zo}$) and `block = 3` ($t = t_{zo} + TI$). At the beginning of the block indexed by `block = 1`, i.e. at time $t = t_{b2}$, we wish to allow Compartment 2 to empty instantaneously into Compartment 1. As we have direct access to the numerical solution to the ODEs (i.e. the `C[.]` array/vector) within the `Adjust` procedure, this is somewhat easier than with the BUGS-language specification. We describe the Compartment 1 adjustment in much the same way (`C[0] := C[0] + C[1]`), but, rather than specifying `C[1] := C[1] - C[1]`, as we have essentially done with the BUGS-language specification, we can write `C[1] := 0` instead.

References

[1] D. J. Lunn, J. Wakefield, A. Thomas, N. Best, and D. Spiegelhalter. *PKBugs User Guide, Version 1.1.* Dept. Epidemiology and Public Health, Imperial College School of Medicine, London, 1999.