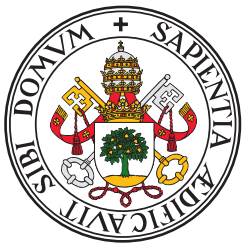

Universidad de Valladolid

Facultad de Ciencias

Departamento de Estadística e Investigación Operativa

TRABAJO FIN DE GRADO:

Algoritmos de estimación de parámetros poblacionales para modelos farmacocinéticos con EcosimPro

Ismael Frutos Barcenilla
Grado en Estadística

Tutelado por:
Dr. Diego García Álvarez
Dr. Agustín Mayo Íscar

Índice

Resumen

La farmacocinética poblacional pretende caracterizar el comportamiento de los fármacos dentro de una población, con el objetivo de aumentar la seguridad y eficacia terapéutica de los fármacos. La modelización de los estudios poblacionales se realiza por medio de modelos mixtos, donde la estimación de máxima verosimilitud no siempre puede obtenerse de manera explícita. Por lo cual se han desarrollado algoritmos para aproximar al máximo verosímil. En este trabajo se exponen diferentes algoritmos clásicos de estimación poblacional utilizados por el proyecto PhysPK de Empresarios Agrupados, dos etapas estándar, FO y FOCE, y además se define para su posterior programación en EcosimPro el algoritmo SAEM, un algoritmo que está basado en una aproximación estocástica del proceso EM. Se resolverá un problema de estimación en un estudio farmacocinético poblacional con los distintos algoritmos explicados comparando resultados de forma conjunta.

Palabras clave: farmacocinética, EcosimPro, PhysPK, SAEM, Metropolis-Hastings.

Abstract

Population-based pharmacokinetic aims to define the pharmaceuticals results in the population, targeting therapeutic assurance and efficiency. The population modeling is performed through mixed models, where maximum likelihood estimation might not always be obtained explicitly. Therefore, algorithms to approach maximum likelihood have been developed. This paper presents different classic algorithms for population estimation used in the PhysPK project of Empresarios Agrupados, two stage method, FO and FOCE, as well as SAEM algorithm definition for its later programming in EcosimPro, which is based in stochastic approximation EM algorithm. An estimation problem in a pharmacokinetic study will be resolved with the different algorithms explained while comparing results altogether.

Keywords: pharmacokinetic, EcosimPro, PhysPK, SAEM, Metropolis-Hastings.

Introducción

El presente trabajo se enmarca en un proyecto del departamento de simulación de Empresarios Agrupados, **PhysPK**, que permite modelar y simular componentes orgánicos con el objetivo de poder representar y resolver problemas farmacológicos. Esta herramienta, que consiste en una serie de librerías **EcosimPro**, plataforma con la que se ha realizado este proyecto, supone un salto cualitativo importante en los campos de biofarmacia y farmacocinética, pues ofrece una solución y basada en simulación, a problemas de gran complejidad y de los que es muy difícil obtener datos.

Esta herramienta se orienta principalmente hacia el campo de la farmacocinética, es importante introducir este concepto que se define como la disciplina de la farmacología que estudia la variabilidad en las concentraciones de fármaco en sangre entre pacientes o voluntarios sanos que reciben un régimen de dosificación estándar con el objetivo de diseñar regímenes de dosis que maximicen la respuesta terapéutica de un fármaco a la par que minimicen los posibles efectos adversos.

Los análisis farmacocinéticos poblacionales son de gran utilidad gracias a que son capaces de caracterizar el comportamiento de un fármaco en poblaciones especiales en las que, por consideraciones éticas o médicas, las muestras que pueden obtenerse son limitadas.

Por lo general, los estudios farmacocinéticos poblacionales utilizan modelos no lineales de efectos mixtos que permiten incluir efectos capaces de describir la variabilidad de los parámetros individuales dentro de la población. El estimador que se desearía usar en estos modelos es el estimador máximo verosímil pero su obtención no es explícita debido a la no linealidad. Por este motivo, cuando el modelo carece de la posibilidad de calcular su función de verosimilitud surge la necesidad de desarrollar métodos o algoritmos que aproximen este valor de la verosimilitud.

En el entorno que se ha trabajado, EcosimPro, están implementados

varios de los algoritmos más usados tradicionalmente para esta estimación, el método de las dos etapas y los llamados métodos de primer orden, FO y FOCE, los cuales linealizan el modelo. El problema de estos métodos es su coste computacional, la necesidad de software de optimización externo a EcosimPro en alguno de sus pasos, el error en la estimación al realizar la transformación del modelo no lineal en uno lineal, encapsulamiento en mínimos locales y los problemas matemáticos que surgen al aumentar la complejidad del modelo. Actualmente, existen algoritmos basados en aproximación estocástica del máximo verosímil que parecen no presentar muchos de estos problemas. Se ha decidido implementar el algoritmo estocástico más extendido actualmente, el algoritmo SAEM.

Objetivo

El objetivo principales de este trabajo es implementar un algoritmo SAEM en EcosimPro para el proyecto PhysPK.

Estructura del trabajo

La memoria de este trabajo se divide en tres bloques:

El primer bloque estaría compuesto por los dos primeros capítulos y tiene un carácter teórico. En este bloque se definen: los conceptos básicos generales de farmacocinética, el entorno de la herramienta utilizada durante todo el trabajo, EcosimPro, y la notación referente a los modelos que se van a tratar.

El segundo corresponde al tercer capítulo donde se introducen los métodos clásicos de estimación poblacional implementados en el proyecto de PhysPK.

Y el tercer y último bloque está formado por los capítulos 4 y 5 que consistirá en la explicación del algoritmo SAEM, su implementación en PhysPK y un estudio comparativo de resultados de los algoritmos de estimación en PhysPK.

Capítulo 1

Farmacocinética en EcosimPro

EcosimPro es una herramienta de simulación de sistemas dinámicos desarrollada por Empresarios Agrupados Internacional S.A. (EAI) que permite modelar diferentes procesos físicos, simples o complejos (EcosimPro (2016)).

EcosimPro permite modelar y simular cualquier fenómeno que se pueda representar mediante ecuaciones algebraico-diferenciales. Para el modelado se ha creado un lenguaje, que toma ideas de otros lenguajes orientados a objetos de uso común hoy en día, llamado **EL** (EcosimPro Language). Es un lenguaje intuitivo pero de gran potencia capaz de desarrollar y modelar tanto procesos continuos como discretos.

El uso de un paradigma de orientación al objeto se debe a que permite un modelado más *natural* de la realidad, facilita la reutilización de componentes de software y ofrece la abstracción como herramienta para sistemas complejos.

La orientación a objetos pretende modelar los sistemas de una forma parecida a cómo percibimos la realidad. Intenta modelar objetos reales con estructuras de software, llamadas *objetos*. Cada uno de estos objetos de software, está compuesto por una serie de características (llamadas *atributos*) y una serie de acciones (*métodos*), al igual que un objeto de la vida real.

1. Farmacocinética

En esta sección se presentan los conceptos teóricos básicos relativos al campo de la farmacocinética que ayudarán a comprender mejor las siguientes secciones del trabajo.

La farmacocinética se define *como la disciplina encargada del estudio de los procesos de absorción, distribución, metabolismo y excreción (ADME) de los medicamentos, de sus bases fisiopatológicas y de sus implicaciones tanto en el diseño de nuevos fármacos y formas farmacéuticas como en la optimización de los tratamientos farmacológicos* (Bauer (2002); Doménech Berrozpe et al. (2013)). En adelante el término de eliminación hará referencia a la combinación de los procesos de metabolismo y excreción, ya que ambos se dan a la vez en el organismo.

Por otra parte, es de interés mencionar que a la relación entre la concentración del fármaco y su respuesta farmacológica se la conoce como farmacodinámica. Es sabido que el cambio en el efecto de un fármaco no suele ser proporcional al cambio en la dosis o en la concentración del fármaco.

Los fármacos deben alcanzar la circulación sistémica para poder ejercer su efecto terapéutico. La ruta de entrada al torrente sanguíneo depende de la vía de administración utilizada. En la administración intravasal el fármaco ingresa directamente en la circulación, mientras que en las extravasales el fármaco deberá disolverse y absorberse para entrar al sistema circulatorio. Además, es importante puntualizar que la absorción, distribución y eliminación de los fármacos ocurre siempre atravesando membranas celulares.

1.1. Farmacocinética poblacional

La farmacocinética poblacional se define *como el estudio de la variabilidad en las concentraciones de fármaco entre pacientes o voluntarios sanos que reciben un régimen de dosificación estándar*. Esta modelización permite estudiar la variabilidad dentro de la población incluyendo efectos relacionados con variables explicativas, como características demográficas (peso, sexo, etc.), ambientales (dietas, tabaco, etc.), fisiopatológicas (embarazo, funcionalidad renal, hepática, etc.) o terapéuticas que influyen en la variabilidad inter-individual (Doménech Berrozpe et al. (2013)).

El análisis farmacocinético poblacional pretende modelizar las observaciones experimentales mediante distribuciones de los parámetros individua-

les que puedan incluir las posibles covariables.

En cualquier estudio farmacológico disminuir la varianza no explicada podría ser importante para aumentar la seguridad y eficacia de un fármaco. Poder explicar la variabilidad en la concentración del fármaco mediante uno o varios de los factores expuestos anteriormente contribuirá a optimizar los regímenes de dosificación a nivel individual o poblacional.

1.2. Modelos farmacocinéticos

Los modelos farmacocinéticos pretenden representar un organismo (humano o animal) por una serie de compartimentos (Davidian (2010)). Bajo una específica ruta de administración de fármaco, la representación compartimental lleva a un sistema de ecuaciones diferenciales, construidas a partir de los parámetros relacionados con los procesos ADME, que describen matemáticamente los ratios de cambio en las cantidades o concentraciones de fármaco que hay en cada compartimento basandose en suposiciones acerca del movimiento del fármaco entre compartimentos.

El caso ideal sería que cada componente del modelo representase un órgano real de un cuerpo, pero los modelos aumentarían mucho más su complejidad, por este motivo, en farmacocinética se tiende a simplificar la representación fisiológica y representar el organismo con un número de compartimentos reducido, entre uno y cuatro. En la figura 1.1 se presenta un ejemplo de sistema de tres compartimentos separados por membranas (TriPool) diseñado con EcosimPro.

Los parámetros farmacocinéticos más importantes en los modelos con los que se va a trabajar en PhysPK son:

- **Aclaramiento** (Cl). Es el parámetro farmacocinético mas importante porque determina el mantenimiento de la dosis que se requiere para obtener el estado de concentración en sangre determinado. Si se conoce el aclaramiento de un fármaco, y se quiere conseguir un determinado estado estacionario de concentración sérica, sería sencillo calcular las dosis necesarias.
- **Volumen de distribución** (V). Es un volumen del compartimento hipotético que relaciona las concentraciones en sangre del fármaco con la cantidad de fármaco en el cuerpo.
- **Ratio de absorción** (k_a). Es la velocidad con que el fármaco entra en el organismo.

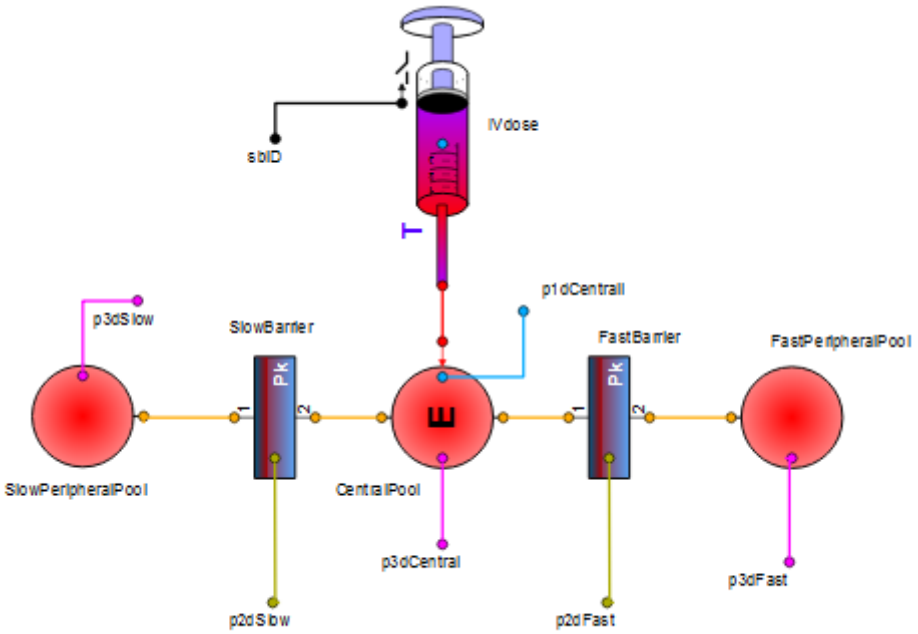

Figura 1.1: Modelo TriPool en EcosimPro

- **Flujos** (Q_*). Mide el flujo que existe entre los distintos compartimentos que forman el modelo farmacocinético, fuertemente relacionado con el tipo de membrana que se deba atravesar entre compartimentos.

Una función típica de ajuste para un modelo que represente la medida de las concentraciones de fármaco en un paciente i podría escribirse como:

$$\begin{aligned} y_{i,j} &= C(t) + \epsilon_{i,j}, \\ C(t) &= \frac{k_a D}{V k_a - Cl}\Big\{exp(-(Cl/V)t) - exp(-k_a t)\Big\}, \end{aligned} \tag{1.1}$$

donde $C(t)$ es una función no lineal de algunos de los parámetros presentados (k_a, V, Cl), de la dosis D y que depende del tiempo.

2. PhysPK

En esta sección se va a describir la metodología general para construir los diferentes modelos en PhysPK dependiendo en su aplicación final (individual o poblacional). Usando los componentes presentes en las librerías que conforman PhysPK es posible simular multitud de modelos.

Con el fin de comprender algunos términos del lenguaje (esquemático, partición, etc.), se ha realizado un apéndice que contiene los elementos básicos de EcosimPro.

A la hora de diseñar un sistema farmacocinético en EcosimPro, existen tres tipos de modelos: base, de entorno y poblacional.

2.1. Modelo base

En PhysPK el modelo base describe la fisiología o sistema farmacocinético necesario para el estudio, es decir, se define el número de compartimentos que tendrá el modelo, el tipo de membranas que separan estos compartimentos y el modo de administración del fármaco.

La construcción del modelo comienza desde el esquemático ya que es aquí donde los sistemas farmacocinéticos pueden diseñarse de manera gráfica. Los componentes que constituyen el modelo se colocan, arrastrando desde una paleta, y conectan, además de definir las especies químicas que deben considerarse.

La figura 1.2 muestra un ejemplo de sistema farmacocinético formado por un único compartimento.

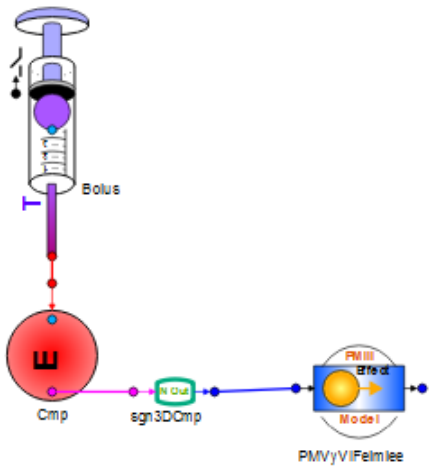

Figura 1.2: Ejemplo de esquemático de un modelo base en EL

Como este modelo será la base para los modelos de entorno y poblacionales, algunas variables (concentraciones, presiones o fluidos) deben ser visibles para futuros modelos que vayan a utilizar el modelo base. La ma-

nera de acceder a las variables requeridas es añadiendo al esquemático dos tipos de puertos diferentes, un primer tipo de puerto que provea acceso automático a concentraciones del componente y otro tipo que provea acceso a variables asociadas a una región concreta dentro del objeto que representa.

Finalmente, algunas variables constantes son definidas en el código del modelo base. Estos datos serán visibles en los modelos de aplicación y parametrizados por el usuario.

2.2. Modelos de entorno

Una vez que el modelo base se ha creado y compilado, se pueden crear infinidad de modelos diferentes dependiendo de las características de la administración del fármaco, su uso o de sus objetivos clínicos o de investigación.

El modelo de entorno añade nuevos elementos de modelado y se conectan a los puertos especificados en el modelo base. Este modelo se compila y crea una partición matemática. Una vez construida la partición, el usuario podrá definir estudios por medio de la simulación de experimentos.

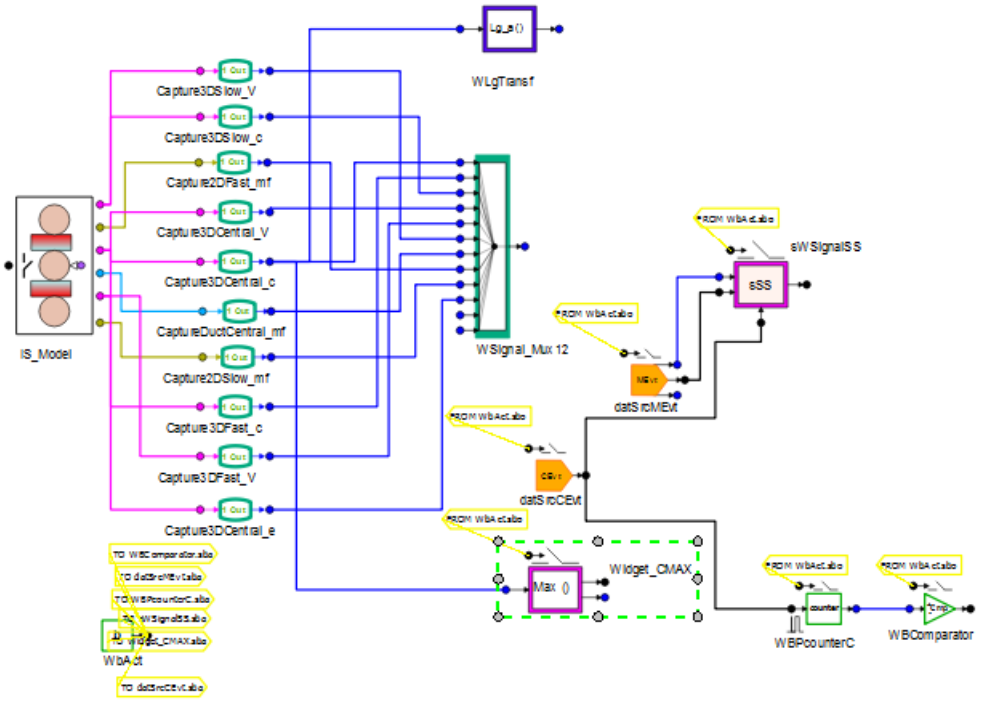

Figura 1.3: Ejemplo de esquemático modelo poblacional en EL

2.3. Modelos poblacionales

Los modelos poblacionales son un tipo de modelos de aplicación centrados en grupos de individuos. Al igual que los modelos de entorno, los poblacionales permiten personalizar las unidades y variables, añadir nuevos elementos, y definir el camino para representar y dirigir la simulación, pero además, añaden una característica que los separa de los modelos de entorno, la posibilidad de que las variables sean de origen aleatorio. Esta cualidad permite que los parámetros varíen de un individuo a otro dentro de la población permitiendo la representación de modelos que contengan efectos aleatorios. Estas modelizaciones poblacionales se realizan por medio de modelos mixtos.

En la figura 1.3 se muestra el esquemático correspondiente a un modelo poblacional.

Capítulo 2

Modelos no lineales de efectos mixtos

Como ya se ha mencionado anteriormente, los modelos farmacocinéticos poblacionales típicamente se modelizan con modelos no lineales de efectos mixtos (Davidian (2010)). Un modelo básico dentro de esta familia vendría dado por la siguiente forma jerárquica:

$$y_{i,j} = f\left(\phi_i, t_{i,j}\right) + \epsilon_{i,j} \tag{2.1}$$

$$\phi_i = g\left(\theta, \mathbf{x}_i\right) + \eta_i \tag{2.2}$$

donde,

- $y_{i,j}$ son las observaciones (normalmente la concentración de fármaco) medidas en el paciente i en el instante j, donde $i = 1$ hasta n, siendo n el número de pacientes considerados, y $j = 1$ hasta m_i, siendo m_i el número de muestras en el tiempo del paciente i.
- $f(\cdot)$ es la función que define la estructura del modelo estimado por PhysPK. Una función no lineal que representa el modelo farmacocinético construido en EcosimPro de manera gráfica mediante un esquemático. $\hat{y}_{i,j} = f(\cdot)$.
- ϕ_i es el vector d-dimensional de parámetros farmacocinéticos asociados con al paciente i-ésimo. $\phi_i = \{\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,d}\}$.

- $t_{i,j}$ es el valor del tiempo en el instante j para el paciente i.
- $\epsilon_{i,j}$ son los errores residuales. Este error incluye la variabilidad intra-individuo. Como norma general, los valores de $\epsilon_{i,j}$ se consideran independientes e igualmente distribuidos que siguen una distribución normal de media 0 y varianza σ^2
- $g(\cdot)$ es la función que calcula los parámetros individuales a partir de los parámetros poblacionales y las posibles covariables de cada sujeto i.
- θ son los parámetros poblacionales $\theta = \{\theta_1, \theta_2, \ldots, \theta_q\}$
- $\mathbf{x}_i$ son las covariables asociadas al sujeto i-ésimo.
- η_i son los residuos de cada individuo. Los parámetros ϕ_i son efectos aleatorios, es decir, varían de un individuo a otro. Al igual que $\epsilon_{i,j}$, los valores η se suponen independientes igualmente distribuidos siguiendo una distribución normal de media 0 y varianza Ω:

  $$\eta_i \sim \mathcal{N}(0, \Omega)$$

  donde Ω es la matriz de varianzas y covarianzas asociada a la variabilidad inter-individuo.

En farmacocinética poblacional es común realizar diferentes modificaciones al modelo base que hemos representado y que tienen que ver con el comportamiento de los residuos de los individuos.

Capítulo 3

Métodos clásicos de estimación poblacional

Debido a la complejidad de la obtención del estimador de máxima verosimilitud, en estos estudios poblacionales se realizan aproximaciones al máximo verosímil mediante diferentes métodos o algoritmos.

1. Métodos de estimación poblacional en PhysPK

Para poder aproximar la función de máxima verosimilitud, necesaria para estimar los parámetros poblacionales que ajusten el modelo a la población, hay implementados diferentes métodos de estimación poblacional en PhysPK.

La figura 3.1 representa el algoritmo general de estimación de parámetros poblacionales en PhysPK.

1.1. Método Estándar de Dos Etapas

Es un método tradicional de estimación poblacional basado en la estimación preliminar de parámetros individuales y a partir de estos tratar de

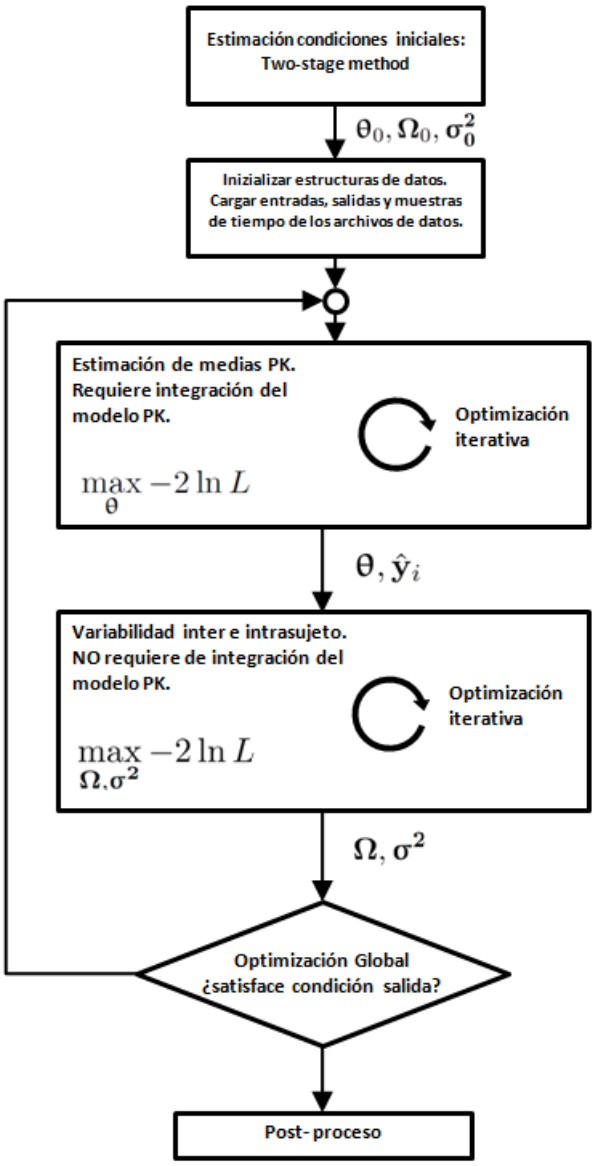

Figura 3.1: Pasos de estimación en PhysPK

determinar los parámetros poblacionales.

En una primera etapa, mediante una rutina externa a EcosimPro (NAG (2012)), se realiza mediante un método SQP (Sequential Quadratic Programming), una estimación de los parámetros farmacocinéticos de cada individuo utilizando sólo los datos de cada uno de ellos. Posteriormente, en una segunda etapa se calcula la media y la varianza de la población a partir

de estos parámetros individuales.

$$\min_\theta \sum_j^{n_i} \Big(y_j - f(\phi, t_j) \Big)^2 \tag{3.1}$$

El principal problema de este método es que, aunque la media poblacional sea razonable, el estimador de la varianza es sesgado, debido a que este método considera que cada paciente es independiente (Davidian (2010)). En PhysPK se utiliza este método para proveer de unos valores iniciales buenos a los otros métodos de estimación poblacional implementados.

1.2. Métodos de primer orden

Esta familia de métodos maximizan la verosimilitud de los parámetros poblacionales θ, Ω y σ^2 para el sujeto i-ésimo, dadas unas muestras y_i y unas desviaciones η_i:

$$L_i \left(\theta, \Omega, \sigma^2 | y_i, \eta_i \right)$$

Estos métodos de estimación son llamados de primer orden porque el modelo es linealizado con relación a la variable η_i usando series de Taylor de primer orden del modelo ($y_{i,j} = f\left(\phi_i, t_{i,j}\right)$):

$$y_{i,j} = h\left(\theta, x_i, t_{i,j}, \eta_{i0}\right) + \left.\frac{\partial h}{\partial \eta_i}\right|_{\eta_{i0}} \eta_i \tag{3.2}$$

con el objetivo de simplificar la estimación del máximo verosímil. Estos métodos son los más usados en los análisis tradicionales de la farmacocinética poblacional.

Los métodos de este tipo implementados en el conjunto de librerías de PhysPK son; FO (First Order) y FOCE (First Order Conditional Estimation) (Wang (2007)).

1.3. FO

En el caso del metodo FO (Wang (2007)), la linearización se realiza entorno al punto $\eta_i = 0$, y la función objetivo asociada a la función de máximo verosímil es:

$$\min_{\theta,\Omega,\boldsymbol{\sigma^2}} -2\ln L = \sum_{i=1}^{n} \left[\ln(\det(\mathbf{C}_i)) + (y_i - \hat{y}_i)^T \mathbf{C}_i^{-1} (y_i - \hat{y}_i) \right],$$

donde,

$$\mathbf{C}_i = \mathbf{G}_i \Omega \mathbf{G}_i^T + \sigma^2 \mathbf{I}_{m_i},$$

siendo $\mathbf{G}_i$ el jacobiano del modelo no lineal, calculado en el punto $\eta_i = 0$,

$$\mathbf{G}_i = \left. \frac{\partial f(\theta, x_i, t_i, \eta_i)}{\partial \eta_i^T} \right|_{\eta_i = 0}.$$

1.4. FOCE

En el caso del método FOCE (Wang (2007)), el modelo se linealiza entorno al punto $\eta_i = \hat{\eta}_i$, donde $\hat{\eta}_i$ se calcula por un método basado en añadir, a priori, una distribución para los parámetros poblacionales.

Este método itera los siguientes dos pasos (Davidian (2010)):

1. Actualizar $\hat{\eta}_i$ con Ω y θ fijos a una estimación actual.
2. Obtener la siguiente estimación de Ω y θ minimizando la ecuación con el valor de $\hat{\eta}_i$ obtenido en el paso anterior:

$$\min_{\theta,\Omega,\sigma^2} -2\ln L =$$

$$= \sum_{i=1}^{n} \left[\ln(\det(\mathbf{C}_i)) + (y_i - \hat{y}_i + \mathbf{G}_i\hat{\eta}_i)^T \mathbf{C}_i^{-1} (y_i - \hat{y}_i + \mathbf{G}_i\hat{\eta}_i) \right],$$

donde los parámetros son idénticos al método FO salvo el jacobiano,

$$\mathbf{G}_i = \left. \frac{\partial f(\theta, x_i, t_i, \eta_i)}{\partial \eta_i^T} \right|_{\eta_i = \hat{\eta}_i} \tag{3.3}$$

El valor inicial de $\hat{\eta}_i$ se calcula por medio del método de dos etapas poblacional, previo a conocer cualquier Ω.

2. Métodos Post-proceso

Una vez estimados los parámetros poblacionales θ, Ω y σ^2, los parámetros individuales para cada sujeto dentro de la población de estudio pueden ser estimados siguiendo el modelo poblacional. En PhysPK este proceso se realiza usando el método Bayesiano citado en FOCE (1.4).

Los parámetros para cada sujeto de la población son $\phi_i = (\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,p})$ y pueden ser estimados minimizando la siguiente función de coste:

$$\min_{\eta_i} \left[\sum_{j=1}^{m_i} \frac{(y_j - \hat{y}_j(\phi_i))^2}{\sigma^2} + (\eta_i)^T \Omega^{-1} (\eta_i) \right] \tag{3.4}$$

Capítulo 4

Algoritmo SAEM

En este capítulo se van a presentar algoritmos encuadrados en la metodología EM, de la que se hablará a continuación, que han despertado recientemente un nuevo interés por su utilidad para la estimación de modelo mixtos. Se incluirá también información acerca de los pasos que se han seguido para su implementación en PhysPK como alternativa a los métodos de primer orden.

1. Algoritmo EM

El algoritmo EM permite estimar por máxima verosimilitud cuando en la matriz de datos tenemos información perdida (Dempster et al. (1977)). Se aplica en situaciones mucho más generales ya que en ocasiones un problema de estimación, donde *a priori* no hay datos perdidos, puede modelizarse como un problema este tipo. Estas modelizaciones alternativas reciben la denominación de datos completos. En ellas, sus correspondientes matrices de datos contienen observaciones que no se han podido medir. El ejemplo más sencillo de este tipo de situación, que motiva la aplicación del algoritmo EM es la estimación máximo verosímil de los parámetros de una mezcla de normales. Este problema se puede representar utilizando un modelo de datos completos que incluye, además de las observaciones de cada individuo, una referencia a la población normal de la que provienen, que es desconocida. Por tanto, tenemos una situación de aplicación del algoritmo debido a estos datos perdidos. Asimismo, en los modelos de efectos mixtos aparecen representaciones que incluyen datos perdidos. Estos corresponderían a los

llamados parámetros individuales, que no son observables, y que podrían considerarse como información perdida.

El algoritmo EM viene dado por la elección de una estimación inicial que iterativamente se mejora con la aplicación de dos pasos, un primer paso-E de obtención de esperanza y un segundo paso-M de maximización. Nombrando a la logverosimilitud correspondiente al modelo de datos completos con

$$L_c(\theta) = \log f(y, z; \theta) \tag{4.1}$$

donde y hace referencia a las observaciones medidas, z a los datos perdidos y θ al vector de parámetros, en el paso-E de la iteración se calcula la función correspondiente a la esperanza de 4.1 dada la estimación actual de los parámetros, θ_{k-1},

$$Q(\theta|\theta_{k-1}) = \mathbb{E}\Big(\log\big(f(y, z; \theta)\big)|y, \theta_{k-1}\Big) \tag{4.2}$$

y en el paso-M se consigue una nueva estimación del vector de parámetros θ_k dada por el punto del espacio paramétrico en el que se alcanza el máximo de la esperanza 4.2 obtenida en el paso-E.

En condiciones de regularidad, es conocido que, en cada iteración del proceso EM aumenta la verosimilitud y que las sucesivas estimaciones convergen a un punto estacionario de la verosimilitud. Como ya se mencionó, el algoritmo EM es candidato natural para la estimación de parámetros en modelos mixtos, pero su uso se ve dificultado cuando la obtención de la esperanza en el paso-E no puede obtenerse explícitamente. Un claro ejemplo de este problema, debido a su complejidad, son los modelos no lineales. Para este tipo de situaciones se han propuesto versiones estocásticas del algoritmo EM que permiten aproximar el paso-E por medio del método de Monte Carlo, como las que se abordan a continuación.

2. Algoritmo SAEM

Para situaciones en las que la obtención de la esperanza del paso-E no es explícita, además de las posibles alternativas basadas en cálculo numérico, existen alternativas estocásticas basadas en simulación. Weim, G. C., & Tanner (1990) propusieron el Monte Carlo EM (MCEM), que sustituye el

paso-E de la iteración k del algoritmo EM por un paso de simulación en el que se obtienen $m(k)$ realizaciones de la distribución $f(z; y, \theta_{k-1})$ por método de Monte Carlo $\{z_k(1), \ldots, z_k(m_k)\}$ para rellenar las observaciones perdidas. Con ellas se construye una estimación de la función $Q(\theta|\theta_{k-1})$, a maximizar en el paso-M, dada por

$$\hat{Q}(\theta|\theta_{k-1}) = \sum_{j=1}^{m_k} \log\big(f(y, z_k(j); \theta)\big). \tag{4.3}$$

Entre las propuestas estocásticas se va a destacar la correspondiente a Delyon et al. (1999), el algoritmo SAEM, que ha motivado una gran cantidad de investigaciones posteriores, muchas de ellas planteadas para dar respuesta a múltiples situaciones que aparecen en la estimación de modelos mixtos. El algoritmo SAEM sustituye el paso-E de la iteración k del algoritmo EM por dos pasos, uno de simulación, equivalente al del algoritmo MCEM, y otro de aproximación, en el que se estima la función $Q(\theta|\theta_{k-1})$ con una media ponderada de la estimación de esta función Q en la iteración anterior $\hat{Q}(\theta|\theta_{k-2})$ y la estimación de la función Q que correspondería a la iteración actual según el MCEM a partir de las realizaciones obtenidas $\sum_{j=1}^{m_k} \log\big(f(y, z_k(j); \theta)\big)$. De esta forma, $\hat{Q}(\theta|\theta_{k-1})$ viene dado por

$$\hat{Q}(\theta|\theta_{k-1}) = (1 - \gamma_k)\hat{Q}(\theta|\theta_{k-2}) + \gamma_k \sum_{j=1}^{m_k} \log\big(f(y, z_k(j); \theta)\big), \tag{4.4}$$

donde γ_k es el valor que determina la ponderación. La sucesión $\{\gamma_k\}_{k\in N}$ que deberá ser elegida *a priori* y estar sujeta a las siguientes condiciones, que garantizan la convergencia del algoritmo:

$$\begin{array}{l} \sum_{k=1}^{\infty} \gamma_k = \infty. \\ \sum_{k=1}^{\infty} \gamma_k^2 < \infty. \end{array} \tag{4.5}$$

Habitualmente se toma una sucesion decreciente con los primeros valores hasta cierta iteracion iguales a 1. De esta forma, en las iteraciones iniciales, el peso de las primeras $m(k)$ realizaciones obtenidas en la iteración k de Monte Carlo será mayor y así la estimación puede variar mucho entre iteraciones pudiendo escapar de máximos locales y, más tarde, los valores de γ_k más pequeños faciliten la convergencia. En implementaciones de SAEM disponibles en la literatura para la estimación de modelos mixtos se utiliza

$m(k) = 1$, o lo que es lo mismo, se realiza una sola simulación en la iteración k, para la obtención de parámetros individuales (Lavielle (2015)).

En modelos mixtos más complejos no es simple obtener realizaciones de las distribuciones condicionadas correspondientes y se obtienen las observaciones del paso de simulación en la iteración k utilizando una sucesión de distribuciones, de cuyos elementos se pueda obtener realizaciones de forma más sencilla, que converja a la distribución condicionada de interés. Para ello están disponibles los métodos como Markov Chain Monte Carlo (MCMC) y, entre ellos, es muy utilizado el algoritmo Metropolis-Hastings (MH).

3. Simulación de datos desconocidos en PhysPK

Se ha implementado SAEM en PhysPK. Para ello se ha utilizado el modelo mixto con varianza residual dado por

$$\begin{aligned} y_{i,j} &= f(\phi_i, t_{i,j}) + \epsilon_{i,j} \\ \phi_i &\sim \mathcal{N}(\beta, \Omega). \end{aligned} \tag{4.6}$$

donde ϕ_i es un vector d-dimensional parámetros individuales, β un vector d-dimensional de los parámetros poblacionales y Ω una matriz de varianzas-covarianzas $d \times d$. Si se supone que la matriz de varianzas-covarianzas Ω es definida positiva, se denota $\theta = (\beta, \Omega, \sigma)$, una descomposición natural de el modelo es,

$$f(\phi, t_{i,j}) = f_1(\phi, t_{i,j}) f_2(\beta, t_{i,j}). \tag{4.7}$$

La implementación de SAEM se realizará siguiendo los pasos de Lavielle (2015) de este modo la iteración k del algoritmo SAEM será:

- **Simulación:** Para $i = 1, 2, ..., N$, simular $\phi_i^{(k)}$ a partir de m iteraciones del algoritmo Metropolis-Hastings con la distribución de los parámetros condicionada al valor de las observaciones, $p(\phi_i | y_i; \mu_{k-1}, \Omega_{k-1})$, como distribución límite.

- **Aproximación estocástica:** Actualizar $s_k = (s_{k,1}, s_{k,2}, s_{k,3})$:

$$s_{k,1} = s_{k-1,1} + \gamma_k \left(\tilde{S}_1(\phi) - s_{k-1,1} \right) \tag{4.8}$$

$$s_{k,2} = s_{k-1,2} + \gamma_k \left(\tilde{S}_2(\phi) - s_{k-1,2} \right). \tag{4.9}$$

$$s_{k,3} = s_{k-1,2} + \gamma_k \left(\tilde{S}_3(y, \phi) - s_{k-1,2} \right). \tag{4.10}$$

  donde, $\tilde{S}(\phi) = (\tilde{S}_1(\phi), \tilde{S}_2(\phi), \tilde{S}_3(\mathbf{y}, \phi))$ es el conjunto de estadísticos suficientes para la estimación del modelo:

$$\tilde{S}_1(\phi) = \sum_{i=1}^{N} \phi_i \tag{4.11}$$

$$\tilde{S}_2(\phi) = \sum_{i=1}^{N} \phi_i \phi_i^t \tag{4.12}$$

$$\tilde{S}_3(\mathbf{y}, \phi) = \sum_{i=1}^{N} \sum_{j=1}^{n_i} (y_{i,j} - f(t_{i,j}, \phi_i))^2. \tag{4.13}$$

- **Maximización:** Con los nuevos s_k, actualizar $(\mu_{k-1}, \Omega_{k-1}, a_{k-1}^2)$:

$$\mu_k = \frac{s_{k,1}}{N} \tag{4.14}$$

$$\Omega_k = \frac{s_{k,2}}{N} - \mu_k \mu_k^t. \tag{4.15}$$

$$a_k^2 = \frac{s_{k,3}}{\sum i = 1^N n_i}. \tag{4.16}$$

Para poder realizar el paso de simulación en PhysPK se ha implementado el algoritmo Metropolis-Hastings siguiendo las indicaciones de Kuhn and Lavielle (2005). Para ello, en el paso de simulación se ha incluido la siguiente rutina para simular los parámetros ϕ_i:

1. Simular $\phi' = (\phi'_1, \ldots, \phi'_n)'$ i.i.d con la distribución, a priori, $\mathcal{N}(\mu_k, \Omega_k)$ y $u = (u_1, \ldots, u_n)$ i.i.d. con la distribución uniforme $\mathcal{U}([0, 1])$.

2. Para $i = 1, \ldots, n$, calcular

$$\begin{aligned}\Delta_i = \sum_{j=1}^{m_i} \Bigg[& \frac{1}{2\sigma^2} \left(y_{i,j} - g(\phi_i', \beta_k, x_{i,j})\right)^2 \\ & - \frac{1}{2\sigma^2} \left(y_{i,j} - g(\phi_i^k, \beta_k, x_{i,j})\right)^2 \Bigg]\end{aligned} \tag{4.17}$$

3. Para $i = 1, \ldots, n$, asignar

$$\begin{aligned}\phi_i^{k+1} &= \phi_i' \Longrightarrow \Delta \leq log(u_i), \\ \phi_i^{k+1} &= \phi_i^k \Longrightarrow resto.\end{aligned} \tag{4.18}$$

En el anexo del trabajo se encuentra el código EL referente a esta implementación.

Capítulo 5

Estudio práctico

Para comprobar el funcionamiento del trabajo realizado se va a crear un problema farmacocinético con PhysPK. El modelo se generará desde cero para evitar utilizar y publicar datos correspondientes a clientes. Con un modelo esquemático con EcosimPro (figura 5.1) que parte del modelo farmacocinético base de tres compartimentos se realizará una estimación de los parámetros poblacionales con cada uno de los métodos de estimación de PhysPK, dos etapas estándar, métodos de primer orden (FO y FOCE) y la implementación de SAEM.

Para la estimación poblacional se uso una base de datos, generada usando PhysPK con unos parámetros poblacionales conocidos, que contiene las concentraciones de un fármaco concreto correspondientes a 50 pacientes, a 15 muestras por paciente, durante las primeras 24 horas desde la administración inicial del fármaco. De las 15 muestras por paciente las 2 primeras serán no válidas al ser tomadas instantes después de la administración de la primera dosis de fármaco. Además, el experimento contará con una segunda administración de dosis de fármaco, en el instante $t = 12$ horas.

El modelo farmacocinético poblacional ajustará las observaciones $y_{i,j}$ por un modelo no lineal de efectos mixtos de la forma:

$$\begin{aligned} y_{i,j} &= f(\phi_i, \beta, t_{i,j}) + \epsilon_{i,j}, \\ \epsilon_{i,j} &\sim \mathcal{N}(0, \sigma^2), \end{aligned} \tag{5.1}$$

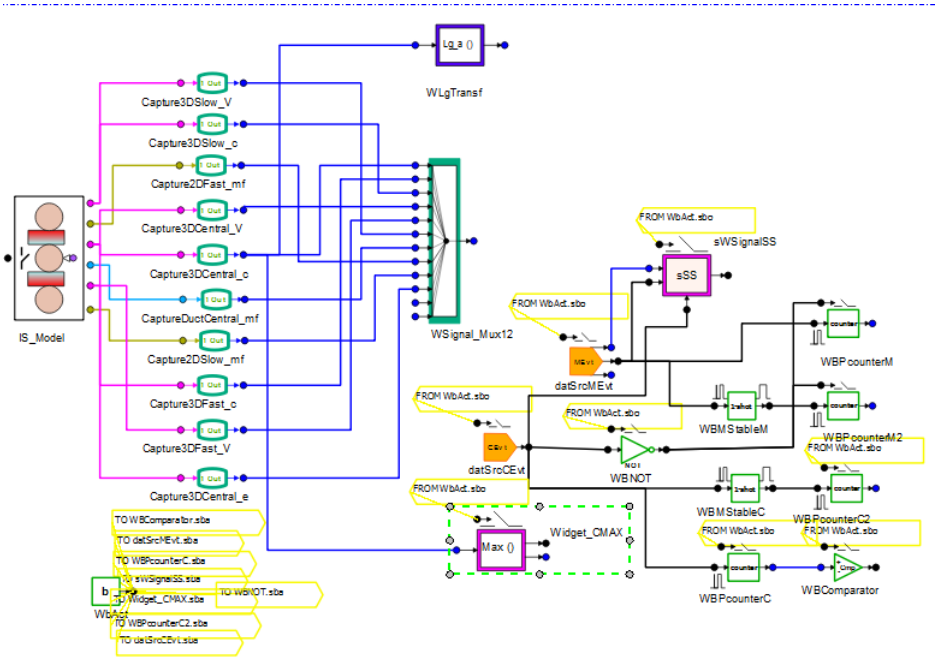

Figura 5.1: Esquemático en EcosimPro del modelo farmacocinético para el ejemplo

$$\begin{aligned} \phi_i &= g(\beta, t_{i,j}) + \eta_i, \\ \eta_i &\sim \mathcal{N}(0, \omega^2), \end{aligned} \tag{5.2}$$

Los parámetros de efectos fijos β en el modelo serán:

- $Vc = 5$ Volumen aparente del compartimento central.
- $Vs = 100$ Volumen aparente del compartimento que refiere al organismo lento.
- $Vf = 17$ Volumen aparente del compartimento que refiere al organismo rápido.
- $Qcf = 44$ Flujo entre el compartimento central y el compartimento rápido.

Estos parámetros β se suponen conocidos en el experimento.

El $t_{i,j}$ toma los valores $\{0{,}1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 18, 24\}$. Los valores en rojo indican las medidas de tiempo para las muestras no válidas.

Los parámetros de efectos aleatorios en el modelo serán el aclaramiento (Cl) y el flujo entre el compartimento central y el compartimento lento (Qcs) que representa los órganos de absorción más lenta del fármaco.

El aclaramiento:

$$Cl_i = \mu_{Cl} e^{\eta_{Cl}},$$

$$\eta_{Cl} \sim \mathcal{N}(0, \omega_{Cl})$$

El coeficiente de flujo lento:

$$Qcs_i = \mu_{Qcs} e^{\eta_{Qcs}},$$

$$\eta_{Qcs} \sim \mathcal{N}(0, \omega_{Qcs})$$

Los parámetros poblacionales a estimar $\theta = (\mu_{Cl}, \mu_{Qcs}, \omega_{Cl}, \omega_{Qcs}, \sigma^2)$

En la tabla 5.1 se muestran de forma conjunta las estimaciones obtenidas para las medias (μ_{Cl} y μ_{Qcs}), la variabilidad inter-individuo (ω_{Cl} y ω_{Qcs}) y la varianza residual (σ^2) del modelo farmacocinético.

	2-Etapas	FO	FOCE	SAEM 10	SAEM 50	Real
μ_{Cl}	19.8682	20.2246	20.0295	19.6128	19.8582	20
μ_{Qcs}	57.6436	58.835	58.9186	76.7712	58.0357	60
$\omega_{Cl_{pop}}$	0.0342	0.0317	0.0325	0.0337	0.0314	0.03
$\omega_{Cl_{pop}}$	0.0625	0.0577	0.0613	0.4865	0.0701	0.05
σ^2	0.0097	0.0102	0.0101	0.0300	0.0099	0.01

Tabla 5.1: Modelo farmacocinético: comparación de estimación de parámetros. El número que sigue a SAEM refiere al número de iteraciones que realizó el algoritmo para esa estimación

Remarcar que las estimaciones en FO y FOCE parten de la estimación ofrecida por el método de las dos etapas, mientras que el algoritmo SAEM comienza con unos parámetros iniciales al azar, alejados de los reales. SAEM fue inicializado con ($\mu_{Cl} = 5$, $\mu_{Qcs} = 15$, $\omega_{Cl_{pop}} = \omega_{Cl_{pop}} = \sigma^2 = 1$) demostrando que es relativamente insensible a los parámetros de arranque del algoritmo.

Conclusiones

Se ha conseguido implementar en PhysPK una primera versión funcional del algoritmo SAEM. Además, el algoritmo realiza buenas estimaciones de los parámetros farmacocinéticos en un reducido número de iteraciones.

Esta implementación permitirá al departamento de simulación de Empresarios Agrupados generar una versión más generalista de estimar parámetros poblacionales en problemas más complejos.

Bibliografía

Bauer, L. a. (2002). *Applied Clinical Pharmacokinetics*, volume 36. Mc Graw Hill.

Davidian, M. (2010). Introduction to Statistical Population Modeling and Analysis for Pharmacokinetic Data. Technical report, Department of Statistics, North Carolina State University.

Delyon, B., Lavielle, M., and Moulines, E. (1999). Convergence of a stochastic approximation version of the EM algorithm. *Annals of statistics*, 27(1):94–128.

Dempster, A., Laird, N., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B Methodological*, 39(1):1–38.

Doménech Berrozpe, J., Martínez Lanao, J., and Peraire Guitart, C., editors (2013). *Tratado General de Biofarmacia y Farmacocinética*. Editorial Sintesis.

EcosimPro (2016). EcosimPro 5.6 User Manual.

Kuhn, E. and Lavielle, M. (2005). Maximum likelihood estimation in nonlinear mixed effects models. *Computational Statistics and Data Analysis*, 49:1020–1038.

Lavielle, M. (2015). *Mixed Effects Models for the Population Approach*. Taylor & Francis Group.

NAG (2012). NAG Library Manual.

Wang, Y. (2007). Derivation of various NONMEM estimation methods. *Journal of Pharmacokinetics and Pharmacodynamics*, 34(5):575–593.

Weim, G. C., & Tanner, M. A. (1990). A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Associatio n , 8 5 (411), 699 - 704 .*

Apéndice

Elementos de un modelo en EcosimPro

Como EcosimPro es una herramienta de simulación con un lenguaje orientado a objetos, es necesario explicar, de manera general, su estructura y conceptos:

- ***WORKSPACE.***

  Es el área de trabajo, consiste en un conjunto de librerías que el usuario utiliza durante el proceso de simulación. Se pueden configurar de distintas formas para organizar mejor el trabajo.

- ***LIBRARY.***

  La librería es un conjunto de elementos que están relacionados en la misma materia. Estos elementos pueden ser: componentes, puertos, funciones, variables globales y constantes. Como es una herramienta multi-disciplinaria, es posible para una librería usar elementos de otra librería, mediante el uso del comando USE.

- ***SCHEMATIC.***

  Es un espacio gráfico donde el usuario puede agregar e instanciar componentes. Se colocan los distintos componentes y se dibujan las relaciones entre ellos como si se tratase de un esquema gráfico del modelo que se pretende construir pero que además cuenta con la posibilidad de modificar y configurar las variables básicas de cada componente utilizado. En definitiva, el esquemático permite la representación gráfica de un modelo topológico y sus propiedades o características.

- ***COMPONENT.***

  Son los elementos básicos de EcosimPro y los más importantes. Guardan un cierto paralelismo con el concepto de clase en Programación

Orientada a Objetos. Un componente representa un sistema o parte del mismo por medio de constantes, variables, ecuaciones algebraico-diferenciales, etc. Los componentes se construyen a partir de 9 bloques opcionales:

- **CONSTRUCTION PARAMETERS.** Bloque para definir y configurar constantes necesarias para el modelo que no deben cambiar durante la simulación del mismo.
- **PORTS.** Permite definir las características de los puertos que podrán conectarse al componente, los puertos utilizados tienen que codificarse por separado. Es la forma de conectar componentes entre si.
- **DATA.** Bloque de definición de variables cuyo valor es conocido, pueden cambiar durante la simulación.
- **DECLS.** Bloque de definición de las variables privadas que describen el comportamiento del componente.
- **OBJECTS.** Bloque destinado a la definición de objetos que son instancias de clases. Se declaran de manera similar a las variables privadas.
- **TOPOLOGY.** Representa de manera textual la instanciación de componentes y la conexión entre ellos, así como su parametrización.
- **INIT.** Mediante una estructura secuencial se define la inicialización de los componentes.
- **DISCRETE.** Parte del componente donde se definen eventos discretos. Para ello se utilizan declaraciones condicionales que ejecutan diferentes acciones cuando las condiciones se cumplan.
- **CONTINUOUS.** Aquí se incluyen las ecuaciones algebraicas y diferenciales necesarias para definir el comportamiento del componente. Estas deben cumplirse a lo largo de la simulación del componente.

```
COMPONENT init_variables(ENUM select_init select=
  component)
  DECLS
    REAL k
    REAL x,z
    REAL y[5]=87
  INIT
    y[3]=5
    IF (select==component) THEN
      x=10
```

```
    END IF
  CONTINUOUS
    EXPAND(i IN 1,5)
    y[i]'=sin(TIME)+x*k'
    x**2=z*x
    z=4*x
    k'=cos(TIME)
END COMPONENT
```

Otro aspecto importante de los componentes es la herencia. Un componente se puede derivar de otro a partir de la directiva (IS_A). Por otra parte, es posible la creación de componentes abstractos (ABSTRACT).

- ***PORT.***

  Los puertos son la única vía de comunicación entre un componente y otro. Permite representar la conexión de dos componentes, añadiendo las ecuaciones de conexión y encapsulando las variables que se intercambian. Además las variables que se encuentran dentro de su estructura son públicas, lo que es de gran utilidad para el modelado de sistemas.

  Deben declararse de manera independiente al comienzo de la librería, para poder incluirlos dentro del cuerpo de cada componente que los implemente.

- ***FUNCTIONS.***

  Las funciones permiten al programador modular los componentes. Todas las variables declaradas dentro de la función son locales, por tanto, sólo son conocidas dentro de la misma función. La motivación principal de la creación de funciones es la posibilidad de la reutilización en distintos componentes sin necesidad de repetir código.

- ***PARTITION.***

  La partición completa el modelo matemático asociado a un componente. Es un paso intermedio entre el componente y el experimento. Para simular un componente, se debe primero definir una partición y después crear experimentos para esa partición.

  Para realizar cualquier simulación con un modelo es necesario convertir el conjunto de ecuaciones escritas por el usuario en un conjunto de ecuaciones escritas de forma secuencial, donde cada línea contenga una variable que se calcula a partir de variables que han sido calculadas en líneas anteriores. De esto se encarga la partición, manipula el modelo para ordenar las ecuaciones y representarlas de una manera resoluble para los algoritmos de integración y las implementa en

un lenguaje causal de C++. Este proceso se denomina *asignación de causalidad* y es el compilador el que decide cual es el orden adecuado para cada ecuación y variable a resolver.

- ***EXPERIMENT.***

  Los experimentos son el elemento ejecutable en EcosimPro, permiten al usuario escribir programas convencionales para simular, por medio de la partición, un componente. En ellos se indica la inicialización de variables dinámicas, la duración de la simulación, etc. En definitiva, en el experimento se concretan las características y condiciones que tendrá nuestro componente durante su simulación.

  En el experimento es donde se realiza la simulación en sí. Se fijan las condiciones iniciales y el entorno. Se establece el paso de integración y valor final para cada variable (u otra información equivalente) sobre la que se realiza la integración. Se puede seleccionar qué variables del modelo mostrar, ya sea gráfica o numéricamente.

Anexo

Script

Algoritmo SAEM

```
/*---------------------------------
 LIBRARY: PHYSPK_POPULATION
 FILE: POP_EST_functionsSAEMDinamic
 CREATION DATE: 25/05/2017
---------------------------------*/
/*******************************
Method: Implements the Bayes block to estimate the eta
  parameters
  It's called from experiment. It calls to
    PostAnalysisMethod function
Parameters
parameterCount: number of population parameters
t1parameterCount: number of t1 parameters
t2parameterCount: number of t2 parameters
*******************************/

USE SYSTEM_EXP_LIB
USE OPTIMIZATION
USE PHYSPK_INTERFACE
USE MATH
USE PHYSPK_WIDGETS
USE PHYSPK_PROCESSES



FUNCTION NO_TYPE simulateModelDinamic(IN EVectorReal
  phiVector[], OUT EVectorReal yModel[], OUT
  EVectorReal yReal[])
```

```
DECLS
  STRING auxString
  REAL TSTART = 0

  BOOLEAN bFlagValidCycle
  BOOLEAN firstInput

  STRING phiNameVector
  INTEGER sampleI
  BOOLEAN bIsOutlier
  BOOLEAN ReachSampleI
  REAL cint
  REAL nextTime
  ENUM t_statusIntegration  intstate  -- Integration
    control
  BOOLEAN bFlagInit = FALSE
  BOOLEAN CaptureDataPerIndividual
  INTEGER t2ParameterCount
OBJECTS
  --jacobian variables
  EVectorString jacobianVariablesNames
  EVectorReal jacobianVariablesDefaultValues

  --decision variables
  EVectorString decisionVariablesNames
  EVectorReal decisionVariablesScaleValues

  dataInputClass currentInput
  dataStudyClass currentStudy
  dataIndividualClass currentIndividual
  plannedInputClass currentPlannedInput

  --boundary variables
  EVectorString boundaryVariablesNames
  EVectorReal boundaryVariablesDefaultValues

  dataCovariateClass currentCovariate
BODY

  t2ParameterCount = estObj.getT2ParameterCount()


  FOR (i IN 1, MAX_INDIVIDUAL_COUNT)
     yModel[i].clear()
     yReal[i].clear()
  END FOR
```

```
FOR (phiI IN 1,phiVector[1].size() )

  estObj.m_bFirstMeasurement = TRUE  -- Assuming
    first valid measurement will be considered
  currentIndividual = estObj.m_data.
    m_individualVector.at(phiI)

  -- Bucle de estudios
  estObj.m_measurementCounter = 0
  FOR (studyI IN 1,currentIndividual.m_studyVector.
    size())

currentStudy = currentIndividual.m_studyVector.at(
  studyI)

-- Init synchronous pulses
estObj.m_data.m_outputDataVectorStatic[1].
  setbsyncMeasurement(FALSE)
-- First border at the beginning
estObj.setBEndCycle(TRUE)
estObj.setTimetoBorder(0)

--Reset the model
setSilentMode(estObj.m_bSilentTS)
RESET_VARIABLES()
setSilentMode(FALSE)

-- Type of experiment
setValueEnum(estObj.getExecPopTypeName(), "
  popEstimation")

--Init Jacobian variables, names given by the user in
   the population experiment
jacobianVariablesNames = estObj.
  getJacobianVariablesNames()
jacobianVariablesDefaultValues = estObj.
  getJacobianVariablesDefaultValues()
FOR (i IN 1, estObj.getJacobianVariablesCount())
  setValueReal(jacobianVariablesNames.at(i),
    jacobianVariablesDefaultValues.at(i))
END FOR

-- Model parameters (DATA)
-- Decision variables
decisionVariablesNames = estObj.
  getDecisionVariableNamesT2()
  -- The following bounds are not used currently but
```

```
    it's possible to used
  -- for controlling access to forbidden regions in
    future
FOR (i IN 1, decisionVariablesNames.size())
  setValueReal(decisionVariablesNames.at(i),phiVector
    [i].at(phiI))
END FOR

-- Set eta values to 0 Eta = 0, considering inter-
  occasion variances
FOR (i IN 1, estObj.getPopParameterCount()+estObj.
  getInterOccasionVarianceCount())
  WRITES(auxString,"Eta[%d]",i)
  setValueReal(auxString,0)
END FOR

--Activated planned flag
FOR (i IN 1,estObj.m_data.m_inputDataVector.size())
  currentInput = estObj.m_data.m_inputDataVector.at(i
    )
  IF (currentInput.m_bDrugActivate) THEN
setValueBool(currentInput.m_planedFlagName,
  currentInput.m_bIsProgrammed)
  END IF
END FOR

-- Set to FALSE all on-line drug administrations
FOR (i IN 1,estObj.m_data.m_inputDataVector.size())
  currentInput = estObj.m_data.m_inputDataVector.at(i
    )
  IF (currentInput.m_bDrugActivate) THEN
IF (currentInput.m_bIsProgrammed) THEN
  setValueBool(currentInput.
    m_activationVariableNameProgCase, FALSE)
  FOR (j IN 1,estObj.getCompoundCount())
FOR (k IN 1,currentInput.
  m_programmedAdministrationCount)
  WRITES(auxString,"%s[Cmp%d,%d]",currentInput.
    m_modelVariableName,j,k)
  setValueReal(auxString,0)
END FOR
  END FOR
ELSE
  setValueBool(currentInput.
    m_activationVariableNameNoProgCase, FALSE)
  FOR (j IN 1,estObj.getCompoundCount())
WRITES(auxString,"%s[Cmp%d,1]",currentInput.
```

```
  m_modelVariableName,j)
setValueReal(auxString,0)
  END FOR
END IF
  END IF
END FOR

TSTART = 0
--Init the model
setSilentMode(estObj.m_bSilentTS)
setValueReal("TIME",TSTART)
EXEC_INIT()
RESET_EVENTS()
setSilentMode(FALSE)

-- First of all
bFlagValidCycle = FALSE
firstInput = TRUE
estObj.m_cyclesNumber = 0
WHILE (NOT bFlagValidCycle)
  -- Control of cycles
  IF NOT estObj.getSimCycleTypeFlag() THEN
bFlagValidCycle = TRUE  -- Only one cycle
  ELSEIF getValueBool(estObj.getSimCycleTypeName())
    THEN
bFlagValidCycle = TRUE  -- This is the last and valid
   cycle
  END IF
  -- Prevalidation variables
  -- Prevalidation variables
  IF NOT estObj.m_bIsOptimizing AND bFlagValidCycle
    THEN
phiNameVector = estObj.getPhiVectorName()
  END IF
  -- Reset measurements number in the current cycle
  estObj.m_cycleMeasurementCounter = 0

  -- Records loop
  sampleI = 1
  WHILE sampleI <= currentStudy.m_outputsValue.rows()
bIsOutlier = currentStudy.m_bIsOutputOutlier.at(
  sampleI,1)
IF NOT firstInput THEN
  ReachSampleI = FALSE
  cint = currentStudy.m_timeVector.at(sampleI) +
    TSTART \
- getValueReal("TIME")
```

```
  -- Adjust cint if plotting and special branch
  IF estObj.m_bExpModelBranch AND estObj.
    m_bPostAnalysisData AND bFlagValidCycle THEN
IF estObj.m_cint_Model < cint - TolTimeInt THEN
  cint = estObj.m_cint_Model
END IF
cint = max(cint, TolTimeInt)
  END IF
  IF cint >= currentStudy.m_timeVector.at(sampleI) +
    TSTART \
- getValueReal("TIME") THEN
ReachSampleI = TRUE  -- It actives once record is
  reached and the it turns off
  END IF

  -- Activate Model for final cycle pulse
  IF (sampleI == 2 AND ReachSampleI) THEN
estObj.setBEndCycle(TRUE)
estObj.setTimetoBorder(currentStudy.m_timeVector.at(
  currentStudy.m_outputsValue.rows())\
  + TSTART - getValueReal("TIME"))  -- TIME will be
    TSTART now
  END IF
  -- Inform Data Src Available Measurement components
  IF ReachSampleI THEN
IF (bFlagValidCycle OR sampleI!=currentStudy.
  m_outputsValue.rows()) THEN
  IF (currentStudy.m_bIsValidMeasurement.at(sampleI
    ,1))  THEN
estObj.m_data.m_outputDataVectorStatic[1].
  setbsyncMeasurement(TRUE)
estObj.m_data.m_outputDataVectorStatic[1].
  setTimetoMeasurement(cint)
gYReal[1] = currentStudy.m_outputsValue.at(sampleI,1)
  END IF
ELSE  -- Take care with transition
  IF (currentStudy.m_bIsValidMeasurement.at(1,1))
    THEN
estObj.m_data.m_outputDataVectorStatic[1].
  setbsyncMeasurement(TRUE)
estObj.m_data.m_outputDataVectorStatic[1].
  setTimetoMeasurement(cint)
gYReal[1] = currentStudy.m_outputsValue.at(1,1)
  ELSEIF (currentStudy.m_bIsValidMeasurement.at(
    sampleI,1)) THEN
estObj.m_data.m_outputDataVectorStatic[1].
  setbsyncMeasurement(TRUE)
```

```
estObj.m_data.m_outputDataVectorStatic[1].
  setTimetoMeasurement(cint)
gYReal[1] = currentStudy.m_outputsValue.at(sampleI,1)
  END IF
END IF
  END IF

  -- Integrate (care with transition)
  nextTime = getValueReal("TIME")+cint
  setSilentMode(estObj.m_bSilentTS)
  intstate = INTEG_TO(nextTime, cint/2)
  -- Check advance
  IF getValueReal("TIME") != nextTime THEN
IF (statePhysModelObj.getModelSuccess()) THEN
  statePhysModelObj.setModelSimExit(Model_Fail,"Place
    unknown")
END IF
WRITE("simulateModel function finishes with errors\n"
  )
RETURN    -- If it is the case, forces to finish to
  the optimizing method
  END IF
  setSilentMode(FALSE)
ELSE
  ReachSampleI = TRUE
END IF

-- Update inputs if record reach
IF ReachSampleI THEN
  --Input variables
  FOR (i IN 1,estObj.m_data.m_inputDataVector.size())
currentInput = estObj.m_data.m_inputDataVector.at(i)
IF (currentInput.m_bDrugActivate) THEN
  IF (currentInput.m_bIsProgrammed) THEN
FOR (k IN 1,currentInput.
  m_programmedAdministrationCount)
  currentPlannedInput = currentStudy.m_plannedInput.
    at(currentInput.m_plannedIndex)
  IF (currentPlannedInput.
    m_plannedAdministrationValue.at(sampleI,k) != 0)
    THEN
    bFlagInit = TRUE
    setValueBool(currentInput.
      m_activationVariableNameProgCase, TRUE)
    WRITES(auxString,"%s[%s,%d]",
  currentInput.m_modelVariableName,estObj.
    getChemicalDrugName(),k)
```

```
      setValueReal(auxString,
    currentPlannedInput.m_plannedAdministrationValue.at
      (sampleI,k))
      WRITES(auxString,"%s[%d]",currentInput.
        m_timeAdmName,k)
      setValueReal(auxString,
    currentPlannedInput.m_plannedAdministrationTime.at(
      sampleI,k))
    END IF
  END FOR
    ELSE
  IF (currentStudy.m_inputNoPlannedMatrix.at(sampleI,
    currentInput.m_noPlannedIndex) != 0) THEN
    bFlagInit = TRUE
    setValueBool(currentInput.
      m_activationVariableNameNoProgCase, TRUE)
    WRITES(auxString,"%s[%s,1]",
      currentInput.m_modelVariableName,estObj.
        getChemicalDrugName())
    setValueReal(auxString,
      currentStudy.m_inputNoPlannedMatrix.at(sampleI,
        currentInput.m_noPlannedIndex))
  END IF
    END IF
  ELSE
    WRITES(auxString,"%s",currentInput.
      m_modelVariableName)
    setValueReal(auxString,
  currentStudy.m_inputNoPlannedMatrix.at(sampleI,
    currentInput.m_noPlannedIndex))
    -- Require init variable
    IF (currentInput.m_bRequireInit) THEN
  IF (sampleI > 1) THEN
    IF (currentStudy.m_inputNoPlannedMatrix.at(sampleI,
      currentInput.m_noPlannedIndex) !=\
     currentStudy.m_inputNoPlannedMatrix.at(sampleI-1,
       currentInput.m_noPlannedIndex)) THEN
      bFlagInit = TRUE
    END IF
  ELSEIF estObj.m_cyclesNumber == 0 THEN
    bFlagInit = TRUE
  ELSEIF (currentStudy.m_inputNoPlannedMatrix.at(
    currentStudy.m_outputsValue.rows(),
    currentInput.m_noPlannedIndex) !=\
       currentStudy.m_inputNoPlannedMatrix.at(1,
         currentInput.m_noPlannedIndex)) THEN
      bFlagInit = TRUE
```

```
END IF
  END IF
END IF
  END FOR
  --Init boundary variables, names given by the user
    in the population experiment
  boundaryVariablesNames = estObj.
    getBoundaryVariablesNames()
  boundaryVariablesDefaultValues = estObj.
    getBoundaryVariablesDefaultValues()
  FOR (i IN 1, estObj.getBoundaryVariablesCount())
setValueReal(boundaryVariablesNames.at(i),
  boundaryVariablesDefaultValues.at(i))
  END FOR
  --occasions
  IF (estObj.getInterOccasionVarianceCount() >0) THEN
setValueInt("currentOccasion",currentStudy.
  m_occasionVector.at(sampleI))
IF (sampleI >1) THEN
  IF (currentStudy.m_occasionVector.at(sampleI) !=\
  currentStudy.m_occasionVector.at(sampleI-1)) THEN
bFlagInit = TRUE
  END IF
ELSEIF estObj.m_cyclesNumber == 0 THEN
  bFlagInit = TRUE
ELSEIF currentStudy.m_occasionVector.at(currentStudy.
  m_outputsValue.rows()) !=\
   currentStudy.m_occasionVector.at(1) THEN
  bFlagInit = TRUE
END IF
  END IF
  --Covariates
  FOR (covariateI IN 1, estObj.m_data.
    m_covariateVector.size())
currentCovariate = estObj.m_data.m_covariateVector.at
  (covariateI)
IF (sampleI > 1) THEN
  IF (currentStudy.m_covariateMatrix.at(sampleI,
    covariateI) != \
   currentStudy.m_covariateMatrix.at(sampleI-1,
     covariateI)) THEN
      bFlagInit = TRUE
setValueReal(currentCovariate.m_modelVariableName,
  currentStudy.m_covariateMatrix.at(sampleI,
    covariateI))
   END IF
ELSEIF estObj.m_cyclesNumber == 0 THEN
```

```
bFlagInit = TRUE
setValueReal(currentCovariate.m_modelVariableName,
  currentStudy.m_covariateMatrix.at(1,covariateI))
ELSEIF (currentStudy.m_covariateMatrix.at(1,
  covariateI) != \
   currentStudy.m_covariateMatrix.at(currentStudy.
     m_covariateMatrix.rows(),
   covariateI)) THEN
     bFlagInit = TRUE
setValueReal(currentCovariate.m_modelVariableName,
  currentStudy.m_covariateMatrix.at(1,covariateI))
END IF
  END FOR
  -- If it is required, the init block is run
IF bFlagInit THEN
  setSilentMode(estObj.m_bSilentTS)
  EXEC_INIT()
  setSilentMode(FALSE)
  bFlagInit = FALSE
END IF
-- Available measurements
IF bFlagValidCycle THEN
  IF NOT bIsOutlier AND NOT firstInput THEN
gYModel[1] = getValueReal(estObj.m_data.
  m_outputDataVectorStatic[1].m_modelVariableName)
gYOut[1] = gYModel[1]     -- To allow branch with
  gYOut
IF (currentStudy.m_bIsValidMeasurement.at(sampleI,1))
    THEN
  estObj.m_cycleMeasurementCounter += 1
  estObj.m_yReal[estObj.m_cycleMeasurementCounter + \
estObj.m_measurementCounter] = gYReal[1]
  estObj.m_yModel[estObj.m_cycleMeasurementCounter +
    \
estObj.m_measurementCounter] = gYModel[1]
  yModel[phiI].append(gYModel[1])
  yReal[phiI].append(gYReal[1])
  --WRITE("%g\t%g\n",estObj.m_yReal[estObj.
    m_cycleMeasurementCounter],estObj.m_yModel[estObj
    .m_cycleMeasurementCounter])

  IF (estObj.getCovariatesFlag()) THEN
FOR (covI IN 1,currentStudy.m_covariateMatrix.cols())
  estObj.m_covariatesCurrentInd[estObj.
    m_cycleMeasurementCounter + \
    estObj.m_measurementCounter,covI] = currentStudy.
      m_covariateMatrix.at(sampleI,covI)
```

```
END FOR
  END IF
  gYOut[1] = gYModel[1]
  -- Report
  IF estObj.m_bPostAnalysisData THEN
REPORT_REFRESH()
  END IF
  -- Validation indices
  IF NOT estObj.m_bIsOptimizing THEN
phiNameVector = estObj.getPhiVectorName()
estObj.m_timeSamplesPerIndividual[estObj.
  m_currentIndividualId].push_back(\
currentStudy.m_timeVector.at(sampleI))
FOR (i IN 1, estObj.getCovariateCount())
  estObj.m_covariatePerIndividual[estObj.
    m_currentIndividualIndex,i].append(\
     currentStudy.m_covariateMatrix.at(sampleI,i))
END FOR
FOR (i IN 1, estObj.getPhiParameterCount())
  WRITES(auxString, "%s[%d]", phiNameVector, i)
  estObj.m_phiValuePerIndividual[estObj.
    m_currentIndividualIndex,i].\
     push_back(getValueReal(auxString))
END FOR
  END IF
END IF
  ELSEIF currentStudy.m_bIsValidMeasurement.at(
    sampleI,1) AND firstInput THEN
estObj.m_bFirstMeasurement = FALSE
  END IF   -- if not bisOutlier
END IF    -- Valid cycle
sampleI += 1
  ELSE  -- ReachSampleI = FALSE
IF estObj.m_bPostAnalysisData AND bFlagValidCycle
  THEN
  gYOut[1] = getValueReal(estObj.m_data.
    m_outputDataVectorStatic[1].m_modelVariableName)
  REPORT_REFRESH()
END IF
  END IF   -- Update inputs for record
  firstInput = FALSE
END WHILE -- Records
  estObj.m_cyclesNumber += 1
  -- Updater TSTART
  IF NOT bFlagValidCycle THEN
TSTART = TSTART + \
  currentStudy.m_timeVector.at(currentStudy.
```

```
      m_timeVector.size())
    END IF

  END WHILE --cycles
    END FOR --studies

  END FOR --individuals

END FUNCTION

//*************************************
// FUNCION DE SIMULACION DE LOS PARAMETROS INDIVIDUALES
//*************************************
FUNCTION NO_TYPE mcmcDinamicC(
  IN INTEGER popParameterCount,
  IN INTEGER t2parameterCount,
  IN REAL mu[t2parameterCount],
  IN REAL omega[popParameterCount,popParameterCount],
  IN REAL sigma2,
  IN INTEGER iter,
  IN INTEGER mcsim,
  OUT EVectorReal individualParamK[],
  OUT EVectorReal gK[]
  )
DECLS
  INTEGER n

  REAL trianguloInd[MAX_INDIVIDUAL_COUNT] = 0

  REAL tauEta=1

  REAL meanMH=0

  BOOLEAN flagResLogAdd=TRUE -- flag scal

OBJECTS

  ERandomVector rln,rn, ru

  EVectorReal individualParamPrima[
    MAX_POPULATION_PARAMETERS]
  EVectorReal y[MAX_INDIVIDUAL_COUNT]
  EVectorReal gPrima[MAX_INDIVIDUAL_COUNT]

BODY

  IF iter<=5  THEN
```

```
  tauEta=1.5
ELSE IF iter<=8 THEN
tauEta=1.3
  ELSE
tauEta=1
  END IF
END IF

FOR (j IN 1, t2parameterCount)
  omega[j,j]=omega[j,j]*tauEta
END FOR

n=estObj.m_data.m_individualVector.size()

FOR (mont IN 1, mcsim)-- bucle para repetir MC

  FOR (j IN 1, t2parameterCount)
individualParamPrima[j].clear()
  END FOR
  FOR (j IN 1, t2parameterCount)
WRITE("\nmu[%d]=%g\tomega[%d,%d]=%g", j,mu[j],j,j,
  omega[j,j])
IF (flagResLogAdd) THEN
  rln.populate(n, DISTR_NORMAL, 0, sqrt(omega[j,j]))
  FOR (i IN 1,rln.size())
individualParamPrima[j].append(mu[j]*exp(rln.at(i)))
  END FOR
ELSE
  rn.populate(n,DISTR_NORMAL, 0, sqrt(omega[j,j]))
  FOR (i IN 1,rn.size())
individualParamPrima[j].append(mu[j]+rn.at(i))
  END FOR
END IF
  END FOR

  ru.populate(n, DISTR_UNIFORM, 0.7, 1)

  simulateModelDinamic(individualParamPrima, gPrima,
    y)

  -- IMPORTANTE: ASEGURAR trianguloInd VACIO
  FOR ( i IN 1, estObj.m_data.m_individualVector.size
    ())
FOR ( j IN 1, y[i].size()) -- j observaciones por
  individuo i
  trianguloInd[i] = trianguloInd[i] + ( ( 1/(2*sigma2
    ) )*(( y[i].at(j) - gPrima[i].at(j) )**2) \
```

```
  -  ( 1/(2*sigma2) )*(( y[i].at(j) - gK[i].at(j) )**2)
    )
  END FOR
    END FOR

    FOR ( i IN 1, estObj.m_data.m_individualVector.size
      ())
  IF ( trianguloInd[i]<=log(ru.at(i)) ) THEN
    FOR ( j IN 1,popParameterCount)
  individualParamK[j].replace(i,individualParamPrima[j
    ].at(i))
    END FOR
    gK[i]=gPrima[i]

  ELSE
    --  individualParamK[j].at(i) = individualParamK[j
      ].at(i)
  END IF

    END FOR
  END FOR

  FOR (i IN 1, 10)
    WRITE("\nID[%d]: ", i)
    FOR (j IN 1, t2parameterCount)
  WRITE("ParamInd[%d]= %g\t", j, individualParamK[j].at
    (i))
    END FOR
  END FOR

END FUNCTION
/*************************************/
/*************************************/
/*************************************/

//************************************
// FUNCION DE APROXIMACION DE LOS PARAMETROS
  POBLACIONALES
//************************************
FUNCTION NO_TYPE aproximationMCMCDinamic(
  IN INTEGER popParameterCount,
  IN EVectorReal individualParamK[],
  IN EVectorReal gK[],
  IN EVectorReal y[],
  IN INTEGER iter,
  OUT REAL statistics[],
  OUT INTEGER samplesCount)
```

```
DECLS
  REAL S1[popParameterCount] = 0
  REAL S1log[popParameterCount] = 0
  REAL S2[popParameterCount] = 0
  REAL S3 = 0

  REAL gamma = 0  -- "Coeficiente de actualizacio" (
    secuencia decreciente de numeros positivos 1/k a
    partir de cierto K)
  INTEGER l=40 -- numero total de iteraciones que
    aproximarÃ¡ con gamma=1

  BOOLEAN flagResLogAdd=TRUE

OBJECTS

BODY
  samplesCount=0

  FOR (k IN 1, popParameterCount)
    FOR (i IN 1, individualParamK[k].size())
  S1[k]=S1[k]+individualParamK[k].at(i)
  IF (flagResLogAdd) THEN
    S1log[k]=S1log[k]+ log_64(individualParamK[k].at(i)
      )
    S2[k]=S2[k]+(log_64(individualParamK[k].at(i)))**2
  ELSE
    S1log[k]=S1log[k]+ (individualParamK[k].at(i))
    S2[k]=S2[k]+(individualParamK[k].at(i))**2
  END IF
  IF (k==1) THEN
    FOR (j IN 1,y[i].size())
  S3=S3+(y[i].at(j)-gK[i].at(j))**2
  samplesCount=samplesCount+1
    END FOR
  END IF
    END FOR
  END FOR

  IF (iter<=l) THEN gamma=1
  ELSE gamma=1/(iter-l)
  END IF

  FOR (j IN 1,popParameterCount)
    -- los primeros j-esimos popParameterCount son las
```

```
      sumas de parametros individuales
    statistics[j]=statistics[j] + gamma*(S1log[j] -
      statistics[j])
    -- a partir de la popParameterCount+1 son las sumas
       de parametros individuales al cuadrado
    statistics[j+popParameterCount]=statistics[j+
      popParameterCount] + gamma*(S2[j] - statistics[j+
      popParameterCount]) -- los sigu
  END FOR
  -- la ultima posicion de statistics seria para el
    error entre observaciones y estimaciones de
    concentracion
    statistics[2*popParameterCount+1]=statistics[2*
      popParameterCount+1] + gamma*(S3 - statistics[2*
      popParameterCount+1])
  -- el verctor statistics contendrÃa los estadisticos
     suficientes para la estimacion poblacional
  -- tendrÃa la siguiente estructura:
  -- EJEMPLO: estatistics[mu1,mu2,mu3,var1,var2,var3,a2
    ] evidentemente solo son sumas, necesitan una
    modificacion para convertirse en medias y var

END FUNCTION

//*************************************
// FUNCION DE MAXIMIZACION DE LOS PARAMETROS
  POBLACIONALES
//*************************************
FUNCTION NO_TYPE maximizationDinamic(
    IN INTEGER popParameterCount,
    IN INTEGER t2parameterCount,
    IN INTEGER samplesCount,
    IN REAL statistics[],
    OUT REAL mu[],
    OUT REAL omega[popParameterCount,popParameterCount
      ],
    OUT REAL sigma2
    )
DECLS
  BOOLEAN flagResLogAdd=TRUE
OBJECTS
BODY

  FOR (j IN 1,t2parameterCount)
    IF (flagResLogAdd) THEN
  mu[j]=statistics[j]/estObj.m_data.m_individualVector.
    size()
```

```
  omega[j,j]= (statistics[j+popParameterCount]/estObj.
    m_data.m_individualVector.size()-(mu[j])**2)
  mu[j]=exp(mu[j])
    ELSE
  mu[j]=statistics[j]/estObj.m_data.m_individualVector.
    size()
  omega[j,j]= (statistics[j+popParameterCount]/estObj.
    m_data.m_individualVector.size()-(mu[j])**2)
    END IF
  END FOR
  sigma2 = statistics[2*popParameterCount+1]/
    samplesCount

END FUNCTION



//************************************
// FUNCION SAEM GENERAL
//************************************

FUNCTION NO_TYPE PC_popSAEMDinamic (
  IN INTEGER popParameterCount,
  IN INTEGER t1parameterCount,
  IN INTEGER t2parameterCount,
  IN INTEGER phiParameterCount)
DECLS
  REAL t1Parameter[t1parameterCount]
  REAL t2Parameter[t2parameterCount]
  REAL t1ParameterScale[t1parameterCount]
  REAL t2ParameterScale[t2parameterCount]
  REAL thetaParameters[t2parameterCount]
  REAL phi[MAX_INDIVIDUAL_COUNT,phiParameterCount]
  INTEGER iter=600
  REAL error=0.0001
  REAL diferencia=1
  INTEGER count=0
  REAL mu[t2parameterCount]
  REAL omega[t2parameterCount,t2parameterCount]
  REAL sigma2
  INTEGER mcsim=1 -- numero de simulaciones de
    montecarlo
  INTEGER samplesCount
  REAL statistics[2*t2parameterCount+1]=0
  REAL ymeans[MAX_INDIVIDUAL_COUNT]
  REAL a,b
  REAL gamma
```

```
  -- Como medida de arrays utilizamos t2parameterCount
    frente a popParameterCount
  -- basandonos en lo que se expone en el articulo E.
    Kuhn, M. Lavielle, 2004
  -- para el tratamiento de poblacionales constantes
    como poblacionales variables evitando asÃ
  -- el NewtonRhapson en la maximizacio

OBJECTS
  EVectorReal individualParamK[
    MAX_POPULATION_PARAMETERS]
  EVectorReal meanValues

  EVectorReal gK[MAX_INDIVIDUAL_COUNT]
  EVectorReal y[MAX_INDIVIDUAL_COUNT]

  ERandomVector rn

BODY
  WRITE("\n Numero de pacientes: %d\n", estObj.m_data.
    m_individualVector.size())

  -- Population parameters inicialization
  -- T2 variables
  meanValues = estObj.getDecisionVariableCIT2()
  FOR (i IN 1, meanValues.size())
    mu[i]=meanValues.at(i)
  END FOR

  -- T1 variables
   -- Initial values for Omega variances
  FOR (i IN 1, t2parameterCount)
    estObj.m_Omega[i,i] = estObj.m_omegaVarCovValue[i,i
      ]
    omega[i,i]=estObj.m_Omega[i,i]
  END FOR

  estObj.m_residualVariance = estObj.
    m_residualVarianceValue
  sigma2=estObj.m_residualVariance

  -- Individual parameters inicialization

  FOR (j IN 1, t2parameterCount)
    individualParamK[j].clear()
  END FOR
  FOR (j IN 1, t2parameterCount)
```

```
  rn.populate(estObj.m_data.m_individualVector.size()
    , DISTR_NORMAL, 0, sqrt(omega[j,j])+sqrt(sigma2))
  FOR (i IN 1,rn.size())
individualParamK[j].append(mu[j]*exp(rn.at(i)))
  END FOR
END FOR
  --IF (count==1) THEN
  simulateModelDinamic(individualParamK, gK, y)
  --END IF

WRITE("\n\t   Initial parameters\n
  ============================== ")
FOR (s IN 1, t2parameterCount)
  WRITE("\n mean[%d]: %g", s, mu[s])
  WRITE("\t var[%d]: %g", s, omega[s,s])
END FOR
WRITE("\n     Varianza residual: %g\n
  ============================== ", sigma2)

-- Start SAEM iterations
WRITE("\n-- COMIENZO ALGORITMO SAEM --\n")
WHILE (count<iter AND diferencia>error OR count<=100)
  count=count+1
  WRITE("\n _=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_=_
    =_=_=_=_=")
  WRITE("\n\t    ITERATION [%d]\n",count)

  -- Simulation_step
  mcmcDinamicC(popParameterCount, t2parameterCount,
    mu,omega,\
  estObj.m_residualVariance, count,  mcsim,
    individualParamK, gK)

  -- Stochastic Aproximation_step
  aproximationMCMCDinamic(popParameterCount,
    individualParamK, gK, y, count, statistics,
    samplesCount)
  WRITE("\n\n Suficient statistics S: \n")
  FOR (s IN 1, 2*t2parameterCount+1)
                      WRITE("\n S[%d]: %g", s,
                        statistics[s])
  END FOR

  -- Maximization_step
  maximizationDinamic(popParameterCount,
    t2parameterCount, samplesCount, statistics, mu,
    omega, sigma2)
```

```
    WRITE("\n\n\t  New Theta: \n
      ==================================== ")
    FOR (s IN 1, t2parameterCount)
                        WRITE("\n mean[%d]: %g", s, mu[
                          s])
                        WRITE("\t var[%d]: %g", s,
                          omega[s,s])
    END FOR
    WRITE("\n     Varianza residual: %g\n
      ==================================== ", sigma2)

    -- diff necesitamos una nueva condicion de salida
      de bucle
    -- diferencia=abs(meanValues.at(1)-mu[1])

    -- Actualizamos los valores dentro del modelo
      ECOSIMPRO
    meanValues.clear()
    FOR (i IN 1,t2parameterCount)
                        meanValues.append(mu[i])
                        estObj.m_omegaVarCovValue[i,i]=
                          omega[i,i]
    END FOR
    estObj.setDecisionVariableCIT2(meanValues)
    estObj.setDecisionVariableScalesT2(meanValues)

    estObj.m_residualVariance=sigma2

  END WHILE

END FUNCTION
```

Experimento

```
/*----------------------------------------
 LIBRARY: PHYSPK_USER
 COMPONENT: TriPoolPKStd_popCL
 PARTITION: default
 EXPERIMENT: estparSAEM
 TEMPLATE: EMPTY EXPERIMENT
 CREATION DATE: 16/04/2017
/*
----------------------------------------
  Taken FROM PHYSPK_POPULATION_template
  CREATION DATE: 03/05/2016
```

```
-------------------------------------*/

// COPY THE FULL CONTENT AND SUBSTITUTE THE FULL CODE
  OF TRANSIENT EXPERIMENT
#include "@PHYSPK_POPULATION@/include/
  PHYSPK_POPULATION_incheaderFOCE.el"

// COVARIATE REGRESSION MODEL (OPTIONAL)
/* Function covariatesModel
 DESCRIPTION: Covariates model for population model
 COMMENTS:
*/
FUNCTION NO_TYPE covariatesModel(IN INTEGER ntheta, IN
  INTEGER nx, IN INTEGER nphi,
  IN REAL vtheta[], IN REAL vx[], OUT REAL vphi[])
BODY
  WRITE("\nEmpty model for non covariate pop model")
END FUNCTION

// ETAS DEPENDENT VALUE FUNCTION (OPTIONAL)
/* Function
 DESCRIPTION: eta dependent value for residual model
 COMMENTS: Real Value = F(etas for a patient)
*/
FUNCTION REAL EtasDependVal(IN INTEGER neta, IN INTEGER
   ntheta, IN REAL veta[],\
  IN REAL vtheta[])
DECLS
  REAL gEtas
BODY
  RETURN 1
END FUNCTION


// SET EXPERIMENT NAME AND MODEL NAME
EXPERIMENT estparSAEM ON TriPoolPKStd_popCL.default
   DECLS
    #include "@PHYSPK_POPULATION@/include/
      PHYSPK_POPULATION_incdecFOCE.el"

    /****************** BEGIN INITIAL CONDITIONS  AND
      BOUNDARIES OF MODEL ***********/
    // Model Initial conditions
    /***  EXTRACT NAMES AND VALUES FROM INIT BLOCK OF
      TRANSIENT EXPERIMENT ****/
    CONST INTEGER OTB_icNumber = 3
    STRING OTB_icName[OTB_icNumber] = {
```

```
                "IS_Model.FastPeripheralPool.Mechanism.
                  Balance.p3d.m[Cmp1]",
                "IS_Model.SlowPeripheralPool.Mechanism.
                  Balance.p3d.m[Cmp1]",
                "IS_Model.CentralPool.MechanismBulk.
                  Balance.p3d.m[Cmp1]"
                }
REAL OTB_icValue[OTB_icNumber] = {
                0,
                0,
                0}

// Model Boundaries
/***  EXTRACT NAMES AND VALUES FROM BOUND BLOCK OF
  TRANSIENT EXPERIMENT ****/
CONST INTEGER OTB_bndNumber = 3
STRING OTB_bndName[OTB_bndNumber] = {
                  "Capture2DFast_mf.p2d.pssur2",
                  "IS_Model.FastPeripheralPool.
                    Mechanism.Balance.p3d.psbulk",
                  "IS_Model.SlowPeripheralPool.
                    Mechanism.Balance.p3d.psbulk"
                  }
REAL    OTB_bndValue[OTB_bndNumber] = {
                1,
                1,
                1}

/****************** END INITIAL CONDITIONS  AND
  BOUNDARIES OF MODEL *****************/

/*************************** BEGIN POPULATION
  STUDY DEFINITION **********************/
// TYPE OF POPULATION ESTIMATION METHOD
ENUM experimentEstTypes experimentEstType = FO
// Name of execution Population type variable
STRING execPopTypeName = "execPopType"
// Type of stochastic model
ENUM residualModelTypes IdResModel = ResAdd

//Samples patient cycles conditioned by an external
   signal
-- It this string is not empty the cycles data are
  conditioned by the boolean variable with that
  value
-- It must refer to a variable of the model
BOOLEAN OTB_PreValid = TRUE      -- Instant of
```

```
  cycle validation
STRING OTB_CycleValidName = ""  -- Name of boolean
  that gives validness to simulation

// FLAGS AND VARIABLES TO CONTROL THE EXPERIMENT
  OBJECTIVES
BOOLEAN EST_ManualMode = FALSE    -- Select manual
  (TRUE) or xml file (FALSE) initial data
BOOLEAN EST_ExpModelBranch = FALSE  -- Branches for
   One Patient mode

BOOLEAN OTB_TwoStage = FALSE
BOOLEAN OTB_POP = FALSE
BOOLEAN OTB_postanalysis = TRUE
BOOLEAN OTB_validationCheck = FALSE
BOOLEAN OTB_OnePatient = FALSE
STRING  OTB_TargetPatidStr  = "1"      -- id
  Patient for individual analysis

-- Maximum Interval time for model plottinh
  communication in timeColumnUnit
REAL EST_CINT_Model = 0.1

// DATA FILES AND DECISION VARIABLES
-- Phi variables
STRING phiVectorName = "phi"

-- Their names can be different from equivalent
  names in models
STRING phiParameterName[phiParameterCount] = {
                              "Ph_CL"
                              }

-- Decision variables t2
-- NOTE: The content of this variable comes from
  the model
STRING OTB_decision_vars_t2_NAME[OTB_ne_t2] = {
                              "Th_CL[Cmp1]"
                              }

-- THETA PARAMETERS AND POPULATION PARAMETERS
  -- T2 (THETA VALUES) (ic for PRE, POP OR input
    data for POST)
REAL OTB_decision_vars_pat_CIT2[OTB_ne_t2] = {40}
  --- IC for t2 var
  -- T1 (POPULATION DISTRIBUTION VALUES) (map for
    PRE, ic for POP OR input data for POST)
```

```
REAL EST_covarianceVal[Nparampop,Nparampop] =
  {{1}}
REAL EST_VarRes = 1  -- ic for POP or input data
  for POST
REAL EST_VarRes2 = 0.3 -- ic for POP or input data
  for POST

  -- Bound and scales for T2 (for PRE OR POP)
REAL OTB_L_est_low_T2[OTB_ne_t2] = {5}  -- lower
  bound for t2 vcar
REAL OTB_L_est_up_T2[OTB_ne_t2] = {100}  -- upper
  bound for t2 var

  -- Bound and scales for T1  (for POP)
REAL EST_covariance_low[Nparampop,Nparampop] =
  {{0.01}}
REAL EST_covariance_up[Nparampop,Nparampop] =
  {{200}}
REAL OTB_VarRes_low = 0.01
REAL OTB_VarRes_up = 100
REAL OTB_VarRes2_low = 0.01
REAL OTB_VarRes2_up = 100

-- INTER-OCASSION VARIABILITY
CONST INTEGER occasionDependedParameterCount = 0
STRING occasionDependedParameter[1] = {""}

// EXPERIMENTAL DATABASE
-- Experimental file
FILEPATH  OTB_expdatafile = "../../
  DVSTriPoolPKpopCLSt1.txt"
-- Column identifications in experiment file
BOOLEAN OTB_bDataFileHasHeader = TRUE
INTEGER OTB_timeCol = 6
STRING timeColumnUnit = "h"
INTEGER OTB_IDsubjectCol = 3
INTEGER OTB_studyCol = 1
INTEGER OTB_occasionCol = 2

CONST INTEGER OTB_outlierCount = 2
INTEGER OTB_outlierVector[OTB_outlierCount] = {1,2}

-- Drug
STRING OTB_chemicalDrug = "Cmp1"

-- inputs DEFINITIONS
CONST INTEGER OTB_inputNumber = 2    -- Number of
```

```
  inputs
STRING OTB_inputNames[OTB_inputNumber] = {
                    "Duration",
                    "IVDrugRate"
                    }
BOOLEAN OTB_severalColumns[OTB_inputNumber] = {
                    FALSE,
                    FALSE   -- TRUE if it is drug
                      planned
                    }
STRING OTB_planedFlagName[OTB_inputNumber] = {
                    "",
                    "p_PrMode"
                    }
INTEGER OTB_inputAdministrationCount[
  OTB_inputNumber] = {
                    0,
                    NAdm
                    }
STRING OTB_inputTimeAdmNames[OTB_inputNumber] = {
                    "",
                    "timeAdm"
                    }
INTEGER OTB_inputCols[OTB_inputNumber] = {
                    5,
                    4
                    }
BOOLEAN OTB_inputReqINIT[OTB_inputNumber] = {
                    FALSE,
                    FALSE
                    }
BOOLEAN OTB_inputDrugAct[OTB_inputNumber] = {
                    FALSE,
                    TRUE
                    }
STRING OTB_actVarNamesProgramCase[OTB_inputNumber]
  = {
                    "",
                    "IOperative"
                    }
STRING OTB_actVarNamesNoProgramCase[OTB_inputNumber
  ] = {
                    "",
                    "IS_Model.sbiD.signal[1]"
                    }

-- outputs DEFINITIONS
```

```
CONST INTEGER OTB_outputNumber = 1    -- Number of
  outputs
STRING OTB_outputNames[OTB_outputNumber] = {
                    "WLgTransf.w_signal[1]"
                    }
INTEGER OTB_outputCols[OTB_outputNumber] = {
                    8
                    }

INTEGER OTB_measCols[OTB_outputNumber] = {
                    9                          }
INTEGER OTB_outlCols[OTB_outputNumber] = {
                    10
                    }

-- covariates DEFINITIONS
STRING OTB_covNames[MAX_COVARIATES_COUNT] = {
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    ""}
INTEGER OTB_covCols[MAX_COVARIATES_COUNT] = {
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0,
                    0}
/***************************** END POPULATION STUDY
   DEFINITION ******************************/


/***************************  BEGIN ADVANCED
  OPTIMIZATION PARAMETERS **********************/

-- Flag that forces that Replicas Data file does
  not exist
```

```
BOOLEAN EST_NoSolutionFile = FALSE

//FLAGS CONTROL OF MODEL ERRORS
BOOLEAN EST_SilentTS = TRUE
BOOLEAN EST_SilentPOP = TRUE
BOOLEAN EST_SilentPOST = TRUE

// CONVERGENCE AND ACCURACY VARIABLES
-- Connvergence main iteration loop FO/FOCE
  estimation method
INTEGER   OTB_iterJ = 10         "maximum iteration
  number in main loop"
REAL      OTB_eps_difJ = 1e-3     "Tolerance to
  finalize the main loop"

 -- Pre-solution variables method (two-stage Method
   )
REAL OTB_REL_ERR_init = 1e-7
REAL OTB_ABS_ERR_init = 1e-7
REAL OTB_f_prec_init = 1e-07    "Optimization
  algorithm precision in two-stage method - 1st"
INTEGER OTB_max_iter_init = 40  "Maximum iteration
  number in two-stage method 1st"
REAL OTB_optim_tol_init = 1e-4  "Accuracy tolerance
   to finalize the two-stage process - 1st"

-- Pre-solution variables method central thetas
  after two-stage
REAL OTB_REL_ERR_initMean = 1e-7
REAL OTB_ABS_ERR_initMean = 1e-7
REAL OTB_f_prec_initMean = 1e-07   "Optimization
  algorithm precision in two-stage method mean
  estimation"
INTEGER OTB_max_iter_initMean = 40   "Maximum
  iteration number in two-stage method 1st mean
  estimation"
REAL OTB_optim_tol_initMean = 1e-2   "Accuracy
  tolerance to finalize the two-stage process - 1st
   mean estimation"


--Accuracy ERROR's in population block
REAL OTB_REL_ERR_pop = 1e-7
REAL OTB_ABS_ERR_pop = 1e-7
-- General Method variables - type 1
  REAL OTB_f_prec_t1 = 1e-07    "optimization
    algorithm precision"
```

```
    INTEGER OTB_max_iter_t1 = 1000   "maximum iteration
      number"
    REAL OTB_optim_tol_t1 = 1e-4     "Accuracy tolerance
       to finalize the t1 process"
    -- General Method variables - type 2
      REAL OTB_f_prec_t2 = 1e-07     "optimization
        algorithm precision"
      INTEGER OTB_max_iter_t2 = 20   "maximum iteration
        number"
    REAL OTB_optim_tol_t2 = 1e-4     "Accuracy tolerance
       to finalize the t2 process"

    -- Method post analysis
    REAL OTB_REL_ERR_post = 1e-8
    REAL OTB_ABS_ERR_post = 1e-8
    REAL OTB_f_prec_post = 1e-8       "Optimization
      algorithm precision for Bayesian process"
    INTEGER OTB_max_iter_post = 40     "Maximum
      iteration number in Bayesian process"
    REAL OTB_optim_tol_post = 1e-3     "Accuracy
      tolerance to finalize the BAYESIAN post-analysis
      process"

    -- Low eigenvalue limit in regularization of
      Covariance Y matrix
    REAL OTB_lowestEigenvalue = 1e-5

    /*************************** END ADVANCED
      OPTIMIZATION PARAMETERS  **********************/

    #include "@PHYSPK_POPULATION@/include/
      PHYSPK_POPULATION_incmainFOCE_decls.el"
    #include "@PHYSPK_POPULATION@/include/
      PHYSPK_POPULATION_incmainSAEM_body.el"


END EXPERIMENT
```