PharmaSUG 2012 - Paper SP03

Computing Initial Values for Pharmacokinetic ML Nonlinear Regression via genetic algorithms and parallel genetic algorithms

Francisco Juretig, Nielsen, Buenos Aires, Argentina

ABSTRACT

In the case of no-random effects the estimation can be carried via nonlinear least squares (proc nlin) or nonlinear maximum likelihood (proc nlmixed). In either case the election of starting values is usually problematic. The derivatives methods will yield an approximate quadratic convergence, as long as the starting values are well chosen. Yet there are two potential problems: if the likelihood is highly irregular, the algorithm will get stuck at local maxima; and if the starting values are too far from the optimum then the algorithm may not converge. If any of these two problems arise, then there are two possible solutions: (1) linearizing the model and estimating it via ordinary least squares. (2) Using a big grid to compute good initial estimates. The first solution entails the problem of potential high correlation because of the improved dimensionality of the problem and an almost certain bias. The second one is probably better, but requires too much computing power. Yet, Genetic algorithms allow a better exploration of the likelihood and provide an improvement per generation. It will be shown how to obtain good initial estimates using a genetic algorithm, especially for the most used pharmacokinetic problems. For nonlinear fixed effects models these values will, usually, yield the maximum of the negative log-likelihood. If it is possible to use a multi-processor computer, it makes sense to parallelize the problem: finally, a macro that can parallelize a genetic maximum likelihood problem into N processors will be presented.

INTRODUCTION

The pharmacokinetic literature is dedicated to understand and predict how drug concentrations evolve in body fluids. This is usually achieved by the use of mathematical models, which state a relationship between certain variables using parameters. These parameters are estimated using actual data, and later this model can be used to compute predictions. The statistical techniques used to solve this problem vary depending on the actual specification of the model. Some of them could specify a linear relationship between the variables, others can specify a non-linear relationship, and some of them may be more focused in understanding how these relationships vary within demographic groups. This last case is studied in the population pharmacokinetics literature using extra random parameters to account for population induced correlation in the data.

Linear models can be estimated via ordinary least squares (such as the REG procedure) or if they have random effects they can be estimated using some modified linear least squares algorithm that accounts for the random effects (such as the GLM procedure and the MIXED procedure). If the model is non-linear the problem can be much harder. The usual approach in SAS® is using the NLIN procedure (nonlinear least squares) or the NLMIXED procedure (maximum likelihood). While the former can only account for fixed effects, the latter is much more powerful since it can handle random effects. Both of them have incorporated very fine-tuned versions of optimizing algorithms, in the case of PROC NLIN the optimization can be carried using Marquandt, Newton Gauss, Steepest Descent, Newton, and Secant method. On the other hand, PROC NLMIXED can use the Trust Region Method, Newton-Raphson with line search, Newton Raphson method with ridging, different versions of quasi-newton methods, various double-dogleg methods, various conjugate gradient methods and the Nelder-Mead algorithm. Generally speaking they will provide a very fast convergence; in particular it will be quadratic for Newton methods, super-linear for quasi-Newton and unknown for Nelder-Mead. The decision on which algorithm to choose is usually based on the dimensionality of the problem, the dataset size, the processor speed and the memory available. In general, Newton methods are to be preferred for small problems, quasi-newton for medium-sized problems (the Hessian is not to be computed, instead it is approximated by the gradient, but much more steps are needed), and conjugate gradient methods are to be preferred for big problems since they do not even need to compute an approximation to the Hessian, thus reducing drastically the amount of memory needed.

All these algorithms need to be provided with initial values for the model parameters. These values are usually chosen either by doing an educated guess based on the structure of the model, or by using estimated parameters from previous studies. Nevertheless, the importance of choosing good initial values cannot be overstated. An incorrect choice of starting values can make the algorithm head towards a plateau, a place where many sets of parameters produce a (very) similar likelihood. In these cases, as the problem becomes very ill identified, numerical problems are deemed to appear. In particular, the inversion of the Hessian that is needed for example in all Newton methods is guaranteed to fail. Secondly, as it has been analyzed, all of the presented algorithms will converge quickly

as long as the initial values are quite “close enough”. This concept is related to the fact that the Newton and Quasi Newton algorithms assume that the function can be well approximated in the vicinity of the optimum point (in the case of maximum likelihood this is the true parameter value) using first and second derivatives. But this leads to an even more severe problem, that those values can be “close enough” to a local but not global maximum. Despite PROC NLMIXED implements advanced tricks in order to push the algorithm towards convergence of a global optimum, such as the OPTCHECK= option that resets the algorithm if it finds a point inside a ball of radius r (default=0.01) with a higher function value than the termination point, convergence to local maxima is frequently to be expected. When any of these problems arise, there are a couple of possible solutions. Firstly, a linearized version using some high degree polynomial can be used, yet this model will not yield neither the same results nor the same interpretation as the non-linear version, and will certainly suffer from a high correlation (for example if variable d has an uniform distribution between 0 and 1, then it has a correlation of 0.97 with d^2). And the matrix inversion problem will now appear also here, since X^T*X will be almost non-invertible. The second solution entails computing a grid of initial values. This is by far, the most common option in most statistical software. This can be achieved in PROC NLMIXED by using the TO - BY option in the PARMS statement. Unluckily, memory and processor requirements will increase exponentially as the amount of grid points or the number of coefficients grows, rendering it useless for many practical problems.

This paper shows how the nonlinear regression problem can be casted into a genetic algorithm optimization problem via PROC GA, and its output can be used as starting values for PROC NLMIXED. These starting values will yield on many occasions practically the maximum likelihood estimates so very few steps, if none, will be needed to converge. Also this approach can be a powerful tool in order to explore the likelihood, thus avoiding local maxima. Additionally, as no derivatives are used, it will be irrelevant if the likelihood is almost flat or nearly discontinuous. This approach is used in 5 methods: the Michaelis-Menten equation, a modified version of the Pinheiro and Bates one-compartment model, a quantification of the reticuloendothelial cell system of the liver model, the four parameter sigmoid-emax problem and the bi-exponential model. Although it is not possible to run a true parallel version of this approach in SAS®, it is shown how this kind of problems can be expressed as a pseudo parallel problem, where the initial values search is split into several programs running concurrently.

GENETIC ALGORITHMS

Over the past years, the reduction of computing prices and the availability of good software implementations has led to an impressive increase in the usage of stochastic search algorithms. These methods share the particular property of using random numbers for finding the maximum/minimum. On most occasions, the solutions that are found using these algorithms cannot be exactly replicated unless the exact same seed (which is used in the random number generators) is used. Most of these algorithms need much less stringent assumptions than their non-stochastic counterparts, or in some occasions, they do not need any assumptions at all. Although these methods can be quite different, a particular subset of these algorithms is built on the idea of replicating how nature optimizes processes. In particular, genetic algorithms are based on the idea that each successive generation of a population can be thought as being the result of an outcome of an optimization process.

The process of building better biological entities can be thought as selecting an initial population of individuals and then choosing usually the fittest of them iteratively. This means that in most occasions the chosen individuals set will consist mostly of individuals which are the best of their generation, but also of fewer individuals from the rest of their respective population. This is of prime importance, and it is probably one of the few ways nature tries to avoid local maxima. If all the chosen individuals were all to be part of the very best group, the population would probably fall quickly into some local maxima, as the selected members of the population would probably be almost identical. The selected population is thus chosen by weighting the genetic diversity versus choosing the best members. After this selection is done, pairs are formed in order to build offspring. Thus, the offspring carry a combination of genetic material of two individuals, with some additional mutations that will render them usually unique. In this fashion, again, the population is ensured to have some diversity. Practically, nature loops through this process *ad-infinitum*.

Genetic algorithms mimic this selection process by assuming that the members of the population are sets of parameters, using the pseudo-random number generator for computing the transition probabilities, evaluating the fit of the solutions by some mathematical function that is to be optimized, and assuming there is a terminal criterion. It is to be noted, that practically no assumption is needed for applying this tool, except for the obvious assumption that this fitness function can be computed for each observation. The usual assumptions made by derivatives based methods: that the function is continuous up to its second derivative, that the hessian can be inverted, or that the initial values are close to the optimal solutions are not needed anymore. Also, it should be noted that it is impossible to state whether this algorithm will effectively attain an optimum and at which rate this convergence may take place.

SAS/OR® GA PROCEDURE

The experimental PROC GA included since SAS/OR® 9.1 allows the user to solve optimization problems using

genetic algorithms. This procedure can include programming statements inside it, such as if/then statements or doing loops, apart from many options to control how the algorithm should behave. Respect to its inner workings, the main point that should be highlighted is that the parameters do not need to be necessarily encoded as bit-strings as in the regular implementations of this algorithm, but they can be encoded as real numbers. In general, the mutation, cross-over and other genetic operators are applied bit-wise to the encoded bit-string, but in the case of PROC GA, they can be applied directly to the real representation of the number. A second important point, is that in the following examples, PROC GA will optimize the log-likelihood, instead of the likelihood (e.g. equation 3), in order to ensure proper comparability with the log-likelihood reported by PROC NLMIXED.

MICHAELIS-MENTEN EQUATION

This well-studied model is used to describe how the reaction rate of enzymes (v_i) relates to the concentration of a substrate (S). As S increases, v_i heads towards V_{max} but never reaches that level. Thus, K_m is of prime importance, as it is the substrate concentration where the reaction rate is half of V_{max} . Assuming $\varepsilon_i \sim N(0, \sigma^2)$ are iid, this two-parameter nonlinear model can be described as:

(1) $v_i = \frac{V_{max} * S_i}{K_m + S_i} + \varepsilon_i$ Equation (3) $l(\theta, x) = \prod_{i=1}^{n} \frac{1}{\sqrt{2*\pi}\sigma} * e^{-0.5*(\frac{v_i - \frac{V_{max}*S_i}{K_m+S_i}}{\sigma})^2}$ with $\theta = (V_{max}, K_m, \sigma)$

In general, there are ways of determining good initial parameters for this model based on the fact that v given by equation 1, is strictly increasing in S. In SAS® (2008, pp. 4267) PROC NLIN documentation, there is a good discussion on how to achieve this; nevertheless a grid that contains those values inside is used there. In most cases, unless the two initial values for the parameters are abnormally chosen and/or the number of observations is really small, the model will generally converge. For reasonably big sample sizes (>30), almost all sets of initial values for the parameters work fine (the models converge quickly to their true values). The same does not hold true when the sample size is less than 30. For instance for n=30, using starting values for Vmax=40 km=40 s2=1.5, PROC NLMIXED returns a "WARNING reporting that the Hessian has at least one negative eigenvalue" and thus the second order optimality conditions are violated (the final estimates are Vmax=172590, km=-121743 and s2=32387). Just for comparison purposes, PROC GA solves that problem easily (final estimates are Vmax=157.922 km=0.0692 and s2=0.70025). When n=1000, both PROC NLMIXED, and PROC GA yield practically the same results; the former produces a negative log-likelihood= 1419.75267 while the latter gives 1419.752668. Estimated coefficients can be verified at Figure 1.

Parameter	true value	PROC NLMIXED	PROC GA
V_{max}	158	158.05	158.04909186
K_m	0.07	0.07371	0.0737115087
σ	1	1.0016	1.0016296117

Figure 1 Simulated Michaelis-Menten equation n= 1000

The SAS® program for generating the simulated data is:

```
1.    data SIMU_DATA_MM;
2.    do day=1 to 1000;
3.    Vmax=158;
4.    Km=0.07;
5.    S=1*day+ranuni(10);
6.    predx =normal(13) + (Vmax*S)/(Km+S);
7.    output;
8.    end;
9.    run;
```

The genetic algorithm is:

```
10.    Proc GA seed = 11 maxiter = 300 data1= SIMU_DATA_MM lastgen=perm.lastgen;
11.    function shubert(selected[*],predx[*],S[*]);
```

```
12.    array x[3]/nosym;
13.    call ReadMember(selected,1,x);
14.    Vmax=x[1];Km=x[2];sd=x[3];
15.     sum1 = 0;
16.         do i=1 to 1000;
17.         gabas = (Vmax*S[i])/(Km+S[i]);
18.         TY=-0.5*(log(2*3.14159265)+log(sd)+((predx[i]-GABAS)**2)/sd);
19.          sum1=sum1+TY;
20.       end;
21.       return(sum1);
22.    endsub;
23.   Call SetEncoding('R3');
24.   array lower[3] /nosym;
25.   array upper[3] /nosym;
26.   lower[1]=0.0;upper[1]=1000;
27.   lower[2]=0.0;upper[2]=500;
28.   lower[3]=0.10;upper[3]=3.1;
29.   call SetBounds(lower,upper);
30.   call SetObjFunc('shubert',1);
31.   call SetCrossProb(0.65);
32.   call SetCross('Heuristic');
33.   call SetMutProb(0.15);
34.   array del[3] /nosym (0.2 0.2 0.2);
35.   call SetMut('Delta','nchange', 1, 'delta',del);
36.   call SetSel('tournament','size', 2);
37.   call SetElite(2);
38.   call Initialize('DEFAULT',400);
39.    Run;
40.     Quit;
41.  proc nlmixed data= SIMU_DATA_MM (drop=Vmax km);
42.  parms Vmax=40 km=40  s2=1.5;
43.  pred=(Vmax*S)/(Km+S);
44.  model predx~ normal(pred,s2);
45.  run;
```

On line 10, the input dataset is specified with the DATA1 statement. In fact more than one data set can be included using DATA2, DATA3, etc. The LASTGEN= option specifies the location of the final generation of parameters. Line 11 defines the function that is to be optimized; in the parenthesis, the SELECTED() array is used for transferring the chosen population of parameters from the previous iteration to the current one; then all the variables of the model are listed. Statement in line 12, tells SAS® to build a three dimensional array that will be used to store the parameters values. In Line 13 CALL READMEMBER instructs SAS® to write the parameters from the previous iteration (stored in SELECTED()) to array x(); these parameters are then used in lines 14-22 to evaluate the log-likelihood in the dataset. Note that every time, PROC GA calls function Shubert, it returns sum1 variable. Line 23 defines the encoding of the problem, which is how SAS® should store the parameters to be optimized and consequently how genetic operators are supposed to be applied to them. In this case, since the problem is three-dimensional, the encoding is R3 (R stands for real). From line 24 to 28, two arrays are created for storing the lower and upper bounds for the parameters. Note that in this case the bounds are quite separated, giving the algorithm a good space exploration. Line 29 specifies that the previous array bounds are to be used for determining the bounds of the problem. Line 30 is straightforward, except for its second argument that specifies whether a maximization or minimization is desired. Line 31 and 32 specify the crossover properties: some of the solutions are passed unchanged to the following generation and some are applied a crossover operation (solution is paired with another solution and offspring is built); then SETCROSSPROB sets how likely this happens; the 'HEURISTIC' option is the only valid option for real valued problems. In line 33, SETMUTPROB indicates how likely it is for a solution to suffer a mutation. Line 35 specifies that the Delta mutation should be used. Essentially it perturbs some components of the solution adding the values in the 'del' array. These values can be fine-tuned according to the problem. Lines 36 and 37 specify that a TOURNAMENT is to be done between pairs of solutions meaning that a competition is done between 2 candidate solutions and the best of them is retained for the next generation. This tries to ensure that certain genetic diversity is present across generations, as the pairs are built at random from the candidate solution set. Tournaments of size=2 place a very weak selective pressure, compared to size=4 or bigger; bigger tournaments imply that the probability that a 'good' solution is present is higher, and thus poor solutions will rapidly disappear after competing with it (but quite a lot of genetic diversity will be lost). Line 37 specifies that always at least 2 of the very best solutions of the generation are passed to the next one (exactly as they are). CALL INITIALIZE at line 38

specifies that a 400 population is to be used across all the optimizations. DEFAULT initialization states that random numbers are to be generated between the bounds and used as initial values.

ONE COMPARTMENT MODELS

These models are extensively used in the literature. Essentially, the body is treated as one single container after some drug is introduced into it, and the drug is assumed to equilibrate rapidly. On more complicated situations these models are used in the study of population pharmacokinetics. In this situation, the idea is to understand *"the sources and correlates of variability in drug concentrations among individuals who are the target patient population receiving clinically relevant doses of a drug of interest"* (FDA 1999, pp. 2). In practical terms, this implies that random effects are to be included in the regression. In this case, obtaining initial values for the numerical optimization will be complicated on most situations. One possible approach here is to try to estimate the model assuming no random effects are present and then use these values for the true model including random effects. One example is the Pinheiro and Bates model for theophylline oral administration used in SAS® (2008, pp. 4403). In this model

$$(2)\quad C_{it}=\frac{D*ke_i*ka_i}{Cl_i*(ka_i-ke_i)}[\exp(-ke_i*t)-\exp(-ka_i*t)]+\varepsilon_{it} \qquad Cl_i=\exp(\beta_1+b_{i1}) \qquad \varepsilon_{it}\sim N(0,\sigma^2)$$

$$ka_i=\exp(\beta_2+b_{i2}) \qquad ke_i=\exp(\beta_3)$$

The β are fixed effects and the b parameters are random effects. The three variables are D (dose), C (concentration of theophylline) and t (time). Doing an educated guess is much harder for this model than for the previous one. One potential solution is to estimate via genetic algorithms this model assuming there are no random effects, and then use these as initial values for the PROC NLMIXED estimation with the random effects. The syntax is:

```
1.    data SIMU_DATA_MM;
2.    do day=1 to 1000;
3.    beta1=-3.22;
4.    beta2=0.47;
5.    beta3=-2.45;
6.    time= 0.03*day+ranuni(13);
7.    dose= 8+3*ranuni(12);
8.    cl    = exp(beta1);
9.    ka    = exp(beta2);
10.   ke    = exp(beta3);
11.   predx =normal(1) + ( dose*ke*ka*(exp(-ke*time)-exp(-ka*time))/cl/(ka-ke));
12.   output;
13.   end;
14.   run;
```

Using PROC GA the following modifications are needed in the previous program used for Michaelis-Menten example.

Line 11 should be replaced for `function shubert(selected[*],predx[*],time[*],dose[*]);`
Line 12 should be replaced for `array x[4]/nosym;`
Line 14 should be replaced for `beta1=x[1];beta2=x[2];beta3=x[3];sd=x[4];`
Line 17 should be replaced for `cl    = exp(beta1); ka    = exp(beta2);    ke    = exp(beta3);`
Line 17 should be replaced for `gabas =(dose[i]*ke*ka*(exp(-ke*time[i])-exp(-ka*time[i]))/cl/(ka-ke));`
Line 23 should be replaced for `Call SetEncoding('R4');`
Line 24 should be replaced for `array lower[4] /nosym;`
Line 25 should be replaced for `array upper[4] /nosym;`
Line 26 should be replaced for `lower[1]=-20.0;upper[1]=-0.1;`
Line 27 should be replaced for `lower[2]=0.0;upper[2]=20;`
Line 28 should be replaced for `lower[3]=-20.0;upper[3]=-0.1;`
Line 28 should be replaced for `lower[4]=0.10;upper[4]=2;`
Line 34 should be replaced for `array del[4] /nosym (0.2 0.2 0.2 0.2);`
Line 38 should be replaced for `call Initialize('DEFAULT',200);`

```
15.   proc nlmixed data= SIMU_DATA_MM (drop=beta1 beta2 beta3);
16.   parms beta1=-3.50 beta2=0 beta3=-3.22 s2=1;
17.   cl   = exp(beta1);   ka   = exp(beta2);   ke   = exp(beta3);
18.   pred= dose*ke*ka*(exp(-ke*time)-exp(-ka*time))/cl/(ka-ke);
19.   model predx~ normal(pred,s2);
20.   run;
```

For this dataset, both the genetic algorithm and PROC NLMIXED without random effects, converge easily to the correct values (Figure 2). Yet, this does not hold when the initial values are not so accurately chosen. In this example, if the initial values are: beta1=-8.5; beta2=8; beta3=-8.22 and s2=1, PROC NLMIXED would not converge correctly.

Parameter	true value	PROC NLMIXED	PROC GA
$\boldsymbol{\beta_1}$	-3.22	-3.2295	-3.2295313
$\boldsymbol{\beta_2}$	0.47	0.4727	0.4726530
$\boldsymbol{\beta_3}$	-2.45	-2.4652	-2.4651538
$\boldsymbol{\sigma}$	1	1.02251	1.0225135

Figure 2 Simulated one compartment model n=1000

QUANTIFICATION OF THE RETICULOENDOTHELIAL CELL SYSTEM OF THE LIVER

This model can be found in Dalgaard (2008). Variable y represents concentration amounts over the liver after a bolus injection of radioactive tracer. Variable t represents time.

(3) $y_i = \beta(1 - e^{-\gamma t_i}) + \varepsilon_i \qquad \varepsilon_i \sim N(0, \sigma^2)$

Data can be simulated using

```
1.  data SIMU_DATA_MM;
2.  do i=1 to 1000;
3.  beta=600;
4.  phi=0.07;
5.  t=i + 5*ranuni(10);
6.  predx =normal(13) + beta*(1-exp(-phi*t)) ;
7.  output;
8.  end;
9.  run;
```

The genetic algorithm needs the following modifications to the base program:

Line 10 should be replaced for: `Proc GA seed = 11 maxiter = 500 data1= SIMU_DATA_MM lastgen=perm.lastgen;`
Line 11 should be replaced for: `function shubert(selected[*],predx[*],t[*]);`
Line 12 should be replaced for: `array x[3]/nosym;`
Line 14 should be replaced for: `beta=x[1];phi=x[2];sd=x[3];`
Line 17 should be replaced for: `gabas =beta*(1-exp(-phi*t[i]));`
Line 26 should be replaced for: `lower[1]=0.0;upper[1]=3200;`
Line 27 should be replaced for: `lower[2]=0.0;upper[2]=3000;`
Line 28 should be replaced for: `lower[3]=0.70;upper[3]=5;`
Line 38 should be replaced for: `call Initialize('DEFAULT',200);`

```
1.    proc nlmixed data= SIMU_DATA_MM(drop=beta phi);
```

```
2.    parms beta=30 phi=20  s2=1;
3.    pred=beta*(1-exp(-phi*t));
4.    model predx~ normal(pred,s2);
5.    run;
```

In this case PROC NLMIXED fails to converge correctly. PROC GA does not have any trouble in selecting the correct parameters. Nevertheless, if the initial values for PROC NLMIXED were beta=600, s2=1.5 and phi=0.05, then it would correctly estimate the parameters (Figure 3).

Parameter	true value	PROC NLMIXED	PROC GA
beta	600	600.05	600.05216385
phi	0.07	0.06995	0.0699466168
$\boldsymbol{\sigma}$	1	1.0010	1.0009851597

Figure 3 RES Model n=1000

PARALLEL GENETIC ALGORITHMS FOR THE 4 PARAMETER SIGMOID E-MAX MODEL

Although PROC GA does not currently support parallelization of genetic algorithm problems, it is possible to build programs that can simulate this behavior. It has been extensively tested how genetic algorithms can be leveraged by multi-processor CPUs. Intuitively, since the algorithm resides essentially on a direct attack on the function to be optimized, many threads may run concurrently and then the best solutions could be gathered together. The possibilities are quite unlimited, for example some threads may search for solutions on different intervals; also some of them could use different cross-over probabilities or simply they could all use different random seeds. In the following example, EMAX_PARAL macro will be discussed. This macro splits an E-max problem into N processors, and submits everything to run concurrently taking full advantage of super-computers or powerful servers. In this example, for simplicity, the macro will split the search into 4 programs each of them having different random seeds and different mutation delta operators (the real numbers that are randomly added to each perturbed solution).

This E-MAX model is probably one of the most important and hardest models in the literature. This model is used for modeling dose-response relationships for a drug. E0 can be thought as the placebo response, EC_{50} is the dose response that produces a 50% of the E_{max} response. C is the concentration or dose of some drug and γ is an additional parameter (known as the Hill parameter). The model and its likelihood can be represented as:

(4) $E_i = E0 + \frac{E_{max} \cdot C_i^{\gamma}}{EC_{50}^{\gamma} + C_i^{\gamma}} + \varepsilon_i \qquad \varepsilon_i \sim N(0, \sigma^2)$ With $l(\theta, x) = \prod_{i=1}^{n} \frac{1}{\sqrt{2*\pi}\sigma} * \mathrm{e}^{-0.5*\left(\frac{x_i - E0 - \frac{E_{max} \cdot C_i^{\gamma}}{EC_{50}^{\gamma} + C_i^{\gamma}}}{\sigma}\right)^2}$

```
1.    %macro rep_data_for_parall(img=);
2.    data perm.data_&img.;
3.    do D=1 to 1000;
4.    Emax=37;
5.    E0=183;
6.    beta=6.22;
7.    ED50=5.06;
8.    dose=80*ranuni(13);
9.    predx =normal(13) + E0 + ((Emax*(Dose**beta))/((ED50**beta) + (Dose**beta)));
10.   output;
11.   end;
12.   run;
13.   %mend;
14.   %macro Emax_paral(threads=);
15.   %do r=1 %to &threads.;
16.   %rep_data_for_parall(img=&r.);
17.   %cast_to_parallel(dataP=data_&r.,seedi=&r.);
```

```
18.   x "runsas parallel&r..sas&";
19.   %end;
20.   %mend;
21.  %macro cast_to_parallel(dataP=,seedi=);
22.  data _null_;
23.        file "./parallel&seedi..sas" dsd dlm="," lrecl=3000;
24. put "Proc GA seed=&seedi. maxiter=1000 data1=perm.&dataP.
lastgen=perm.lastgen_P&seedi.;";
25.   put "function shubert(selected[*],predx[*],Dose[*]);";
26.   put "array x[5]/nosym;";
27.   put "       call ReadMember(selected,1,x);";
28.   put "        E0=x[1];Emax=x[2];ED50=x[3];beta=x[4];sd=x[5];";
29.   put "        sum1 = 0;";
30.   put "        do i=1 to 1000;";
31.   put "   gabas =E0 + ((Emax*(Dose[i]**beta))/((ED50**beta) + (Dose[i]**beta)));";
32.   put "        TY=-0.5*(log(2*3.14159265)+log(sd)+((predx[i]-GABAS)**2)/sd);";
33.   put "        sum1=sum1+TY;";
34.   put "end;";
35.   put " return(sum1);";
36.   put "  endsub;";
37.   put " Call SetEncoding('R5');";
38.   put " array lower[5] /nosym;";
39.   put " array upper[5] /nosym;";
40.   put " lower[1]=0.0;upper[1]=1000;";
41.   put " lower[2]=0.0;upper[2]=100;";
42.   put " lower[3]=0.0;upper[3]=100;";
43.   put " lower[4]=0.10;upper[4]=100;";
44.   put " lower[5]=0.70;upper[5]=2;";
45.   put " call SetBounds(lower,upper);";
46.   put " call SetObjFunc('shubert',1);";
47.   put " call SetCrossProb(0.65);";
48.   put " call SetCross('Heuristic');";
49.   put "call SetMutProb(0.15);";
50.   put " array del[5] /nosym (%sysevalf(&seedi./10) %sysevalf(&seedi./10)
%sysevalf(&seedi./10) 51.%sysevalf(&seedi./10) %sysevalf(&seedi./10));";put "call
SetMut('Delta','nchange', 1, 'delta',del);";
52.  put " call SetSel('tournament','size', 2);";
53.  put " call SetElite(2);";
54.  put " call Initialize('DEFAULT',200);";
55.  put " Run;";
56.  put " Quit;";
57.  run;
58.  %mend;

59.  %Emax_paral(threads=4);
```

On line 18, the x command executes an operating system instruction. In this case, the function “runsas” calls SAS.exe to run the corresponding SAS program.

Parameter	True values	Thread1	Thread2	Thread3	Thread4
Emax	37	37.157741482	37.157741366	37.157741369	37.157741282
E0	183	182.85872113	182.85872124	182.85872126	182.85872129
Beta	6.22	6.0602945314	6.0602945823	6.0602945921	6.0602945341
Ed50	5.06	5.033393557	5.0333935572	5.033393568	5.0333935543
σ	1	0.9169083996	0.9169083981	0.9169083702	0.9169084159

Figure 4 Emax_paral macro threads =4

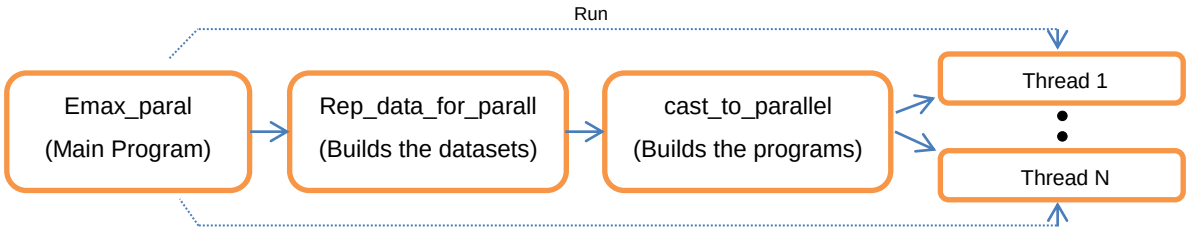

Figure 5 E-MAX_PARAL MACRO

In general, PROC NLMIXED would have found it difficult to estimate this model. For example: if the starting values are e0=500, Beta=0.1, ED50=2.06 and sigma=1, then the model will not converge correctly to the global maximum. Note that in real situations, the difference between different threads can be much greater than in Figure 4.

BI-EXPONENTIAL REGRESSION

Incorporating restrictions into estimation algorithms will, in most practical cases, be difficult. These restrictions are usually included for two reasons: either for incorporating a priori knowledge about some parameter, for example some variable should have a negative/positive impact; or they could be included in order to ensure correct identification of the model. One example of this last situation is when a researcher formulates a bi-exponential model. This model represents the typical open two-compartment model. It has been very used in the literature, for example Rotschafer et al. (1982) fit this model to serum vancomycin concentration data. This model is described by (where variable t stands for time, and y stands for concentration of some drug)

(5) $y_i = A * \exp(-\alpha_1 * t_i) + B * \exp(-\alpha_2 * t_i) + \varepsilon_i \qquad \varepsilon_i \sim N(0, \sigma^2)$

It is readily evident that this model has a problem, since the pairs (α_1, A) and (α_2, B) are exchangeable. There is no way for the model to know where to place each pair. Almost every possible algorithm is expected to oscillate between the two solutions. Bonate (2011, pp. 114) shows how initial values can be obtained by using some algebra arguments and then using an auxiliary regression. For example if data is simulated by:

```
1.   data data_biexp(keep=predx dose);
2.   do D=1 to 1000;
3.   dose=0.1*D+2*ranuni(13);
4.   A=70;
5.   B=50;
6.   alpha1=0.5;
7.   alpha2=0.05;
8.   predx =normal(13) + A*exp(-alpha1*Dose) + B*exp(-alpha2*Dose);
9.   output;
10.  end;
11.  run;
```

In this context the resulting parameters obtained via PROC NLMIXED are expected to be exchanged. For example starting values A=10.818124084, B=5.165860321, alpha1=0.5292495554, alpha2=0.0808308783 s2=1, yield the inverted coefficients although the initial values are reasonably close to the true values. This problem also happens using PROC GA. Thus, the following restriction is added $\alpha_1 > \alpha_2$. In this case the model is completely identified. This restriction can be easily accommodated through the very powerful programming statements inside PROC GA. The core of the program should be modified as follows

Line 10 should be replaced for: `Proc GA seed = 11 maxiter = 1000 data1=data_biexp lastgen=lastgen;`

```
1.  function shubert(selected[*],predx[*],Dose[*]);
2.        array x[5]/nosym;
3.        call ReadMember(selected,1,x);
4.        A=x[1];alpha1=x[2];B=x[3];alpha2=x[4];sd=x[5];
5.        sum1 = 0;
```

```
6.        if alpha1>=alpha2 then do;
7.         do i=1 to 1000;
8.         gabas =A*exp(-alpha1*Dose[i]) + B*exp(-alpha2*Dose[i]);
9.        TY=-0.5*(log(2*3.14159265)+log(sd)+((predx[i]-GABAS)**2)/sd);
10.         sum1=sum1+TY;
11.        End;
12.        end;
13.        else do;
14.       sum1=-50000000;
15.       end;
16.      return(sum1);
17.     endsub;
```

Line 23 should be replaced for: `Call SetEncoding('R5');`
Line 24 should be replaced for: `array lower[5] /nosym;`
Line 25 should be replaced for: `array upper[5] /nosym;`
Line 25 should be replaced for: `lower[1]=0.0;upper[1]=500;`
Line 26 should be replaced for: `lower[2]=0.0;upper[2]=500;`
Line 26 should be replaced for: `lower[3]=0.0;upper[3]=500;`
Line 26 should be replaced for: `lower[4]=0.0;upper[4]=500;`
Line 26 should be replaced for: `lower[5]=0.70;upper[5]=2;`
Line 34 should be replaced for: `array del[5] /nosym (0.2 0.2 0.2 0.2 0.2);`

In this case, if the program detects that alpha1<alpha2 a very large negative likelihood is assigned to the sum1 variable. If, on the contrary, alpha1>=alpha2 then the calculation goes as usual. In this way, hopefully, PROC GA will evade situations where alpha1<alpha2. The output for this program is A=70.032521422, alpha1=0.4957619702, B=49.817540593, beta2=0.0498241655 and sigma=0.9199913639. If these restrictions were not to be included, then the resulting estimates would have been inverted. Still, it seems evident that the smartest way of taking full advantage of PROC NLMIXED and PROC GA is by putting them together to solve an estimation problem. PROC GA can evade most of the local maxima problems, while PROC NLMIXED is equipped with sophisticated derivatives algorithms. This example shows how to fetch the selected values from PROC GA and using them as initial values in PROC NLMIXED automatically. In this case, the model converges in one single step with a gradient slope= -0.00002. (PROC GA outputs a table with rows as generations, being the first row the best one).

```
1.   Data _Null_;
2.       set lastgen;
3.        if _n_=1 then do;
4.                call symputx('A',A1);call symputx('alpha1',A2);
5.                call symputx('B',A3);call symputx('alpha2',A4);
6.                call symputx('sigma',A5);
7.        end;
8.    run;
9.    proc nlmixed data=data_biexp;
10.       parms
11.       A=&A.
12.       B=&B.
13.       alpha1=&alpha1.
14.       alpha2=&alpha2. s2=&sigma.;
15.       pred =A*exp(-alpha1*Dose) + B*exp(-alpha2*Dose);
16.       model predx~ normal(pred,s2);
17.    run;
```

Iteration History

Iter	Calls	NegLogLike	Diff	MaxGrad	Slope
1	26	1377.24303	3.18E-12	0.000386	-0.00002

NOTE: FCONV convergence criterion satisfied.

Figure 6 PROC GA+PROC NLMIXED immediate convergence

CONCLUSION

Experimental SAS/OR® PROC GA can prove very useful in the determination of initial values for nonlinear regression in general and pharmacokinetic modeling in particular. As it has been seen, it can work very well in situations where the traditional determination of the initial values is difficult or infeasible. Also it can accommodate very well hard restrictions such as when the value of certain parameters is expected to be lower/greater than other parameters. Although this method only takes into account the fixed effects, in mixed effects models, this method can be used with the fixed effects part to fetch initial values. Additionally, if a multi-processor computer is available, then this method can become leveraged by splitting the search into multiple processors. In all cases the best approach is to run PROC GA and then read the resulting values from the last generation, as initial values, into PROC NLMIXED.

REFERENCES

- Bonate, Peter (2011). Pharmacokinetic-Pharmacodynamic Modeling and Simulation. New York, NY. Springer. pp. 110,114.
- Dalgaard, Peter. Nonlinear Regression Analysis lecture notes. Department of Biostatistics, University of Copenhagen. May 2008. Available at http://staff.pubhealth.ku.dk/~pd/V+R/handouts/nonlin-2x2.pdf
- Rotschafer, J., Crossley, K., Zaske, D., Mead, K., Sawchuk, R., and Solem, L. (1982), “Pharmacokinetics of Vancomycin: Observations in 28 Patients and Dosage Recommendations”, Antimicrobial Agents and Chemotherapy, 22, pp. 392.
- SAS Institute Inc. (2008). SAS/STAT® 9.2 User’s Guide. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (2004). SAS/OR® 9.1 User’s Guide: Local Search Optimization, Cary, NC: SAS Institute Inc.
- U.S. Department of Health and Human Services, Food and Drug Administration (1999). Guidance for Industry. Population Pharmacokinetics.pp 2. Available at http://www.fda.gov/downloads/ScienceResearch/SpecialTopics/WomensHealthResearch/UCM133184.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Francisco Juretig
Enterprise: Nielsen
Address: Tucumán 348
City, State ZIP: Buenos Aires - Argentina
Work Phone: +54 11 4891 1100
Fax: +54 11 4891 1120
E-mail: Francisco.Juretig@nielsen.com or fjuretig@yahoo.com
Twitter: FJuretig