**Using Count Measures in NPAG**
Pmetrics::NPAG ver122 Tested on Greco/Meropenem
January 10, 2019

**Summary.** All necessary code to process count data is included in NPAG fortran modules. Count data is assumed to emerge from a Poisson distribution with mean equal to the PK model prediction. Data entry conforms to previous Pmetrics csv standard. Users identify observations as count variables (vs. continuous) by setting the respective output error polynomial to -229, 10, -229, -229 [See Pmetrics manual for explanation of error polynomial]. Note: Large count observations (greater than 30) are treated as Normal observations with standard deviation computed using the Gamma function approximation.

Gamma optimization requires comparison of three solutions (using three different values of gamma, while holding support point placement fixed) at each cycle of NPAG – all changes to the code to support count measures are automatically included in the gamma optimization.

Population fixed parameter estimation depends on minimization of $\sum z^2$ for all observations; where, $z = (\text{observation} - \mu)/\sigma$. For optimization of combined count and Normal observation data, $\mu$ and $\sigma$ for count data are now calculated assuming a Poisson distributed variable.

The program is validated on Normal observations using the Meropenem plus Fosphomycin dataset (MEROFOS-01). Poisson ready code, optimized to

-LL = -630.45386940614571, while the Normal-only code optimized to

-LL = -630.45388565747373. A difference of 1.625133e-05, which passes the statistical consistency check ($10^{-4}$). Both programs converged after 846 cycles. Minor differences in the output logs suggest the new program followed a very similar optimization path. Other statistics are in the report, below.

Validation on count data will be done after a suitable simulation dataset is produced. The MEROFOS-01 dataset contains both Normal. and Count observations and will be used as the “real-world” test set.

Three notes: First, observed performance improvement in the code indicate that we still have much work to do to improve the memory model within the parallelized region. Second, the small discrepancy in search path and endpoint, suggest we need to spend effort to carefully construct an appropriate validation dataset. Finally, much work validating NPAG will be necessary as we move to using the proper surrogate of the mean.

**Motivation.** Pmetrics is a R package that includes support for translating data and models into fortran code for nonparametric population PK analysis. The computational engines called by Pmetrics assume observed measurements are Normally distributed. For count data, if counts are small (less than several dozen), a Normal approximation to the measurement distribution can pose a limitation on the veracity of Pmetrics. In this situation, it is desirable for NPAG to assume Poisson distributed measurements.

NPAG is a semi-parametric optimization method, carried out in several steps, summarized as

**START** Propose a set of supports (complete PK model description)

Calculate predicted observations

$\Psi$ Calculate the probability of each observation dependent on each support

$\lambda$ Calculate the probability of each support

Prune the support set to only the most probable supports; increasing likelihood

$\gamma, \Lambda$ Adjust variables that affect the probabilities $\Psi$ and $\lambda$

choose the $\gamma$ and $\Lambda$ that decrease the sum of the square of the normalized distance between observation and prediction

Call a Nelder-Mead optimization to adjust

**Cycle** back to START

The calculations required to handle Normally distributed observations is dispersed throughout the above cycle. Parts of this cycle are parallelized and require careful handling. Including ability to handle count data has proven too complicated a task given limited manpower and time.

**Background.** Pmetrics calls the Fortran program NPAG, which carries out a semi-parametric population analysis of the observed data for the given model. At the heart of this analysis is the calculation of the probability of an observation given a complete set of PK parameters, *a.k.a.* support point, $g$. In PK analysis, observations are multidimensional time series. For dimension $j$ and time of observation $k$

$$p(Y_{obs}|g) = \prod_{j,k} p(Y_{obs;j,k}|g) \tag{1}$$

Each dimension of the observation represents a unique biological measurement, for example concentration of drug or viral load, which is assumed to emerge from a unique statistical distribution. Currently, NPAG assumes these distributions are Normal, but with different functional descriptions of mean and standard deviation: $\mu = F_{j,k}(g)$ and $\sigma_{j,k} = \hat{f}(\mu, \gamma, C)$. [1]

$$p(Y_{obs}|g) = \prod_{j,k} \left(\sigma_{j,k}\sqrt{2\pi}\right)^{-1} \exp\left(\frac{-(Y_{obs;j,k} - F_{j,k}(g))^2}{2\sigma_{j,k}}\right)$$

The current code calculates $p(Y_{obs}|g)$ in three steps:

1 The approximation $\mu = Y_{obs}$ is made, $\sigma_{j,k} = f(Y_{obs}, \gamma, C)$ is calculated and stored, and $s = \prod_{j,k} \left(\sigma_{j,k}\sqrt{2\pi}\right)^{-1}$ is calculated.

[1] $\hat{f}$ is a polynomial function that requires no explanation here. Explanation is given in [].

2 The multidimensional time series $F_{j,k}$ is calculated at each time $k$, and $w = \sum_{j,k} \left( \frac{(Y_{obs;j,k} - F_{j,k}(g))}{\sigma_{j,k}} \right)^2$ is updated. Note: This section of code is parallelized over $g$.

3 $p(Y_{obs}|g) = s \exp(-w/2)$.

**Strategy.** In the above equations, each $j$ implicitly represents a unique distribution. The product in equation 1 can be written

$$p(Y_{obs}|g) = zq \tag{2}$$
$$z = \prod_{j,k} p(Y_{obs;j,k}|g); \text{for } j \text{ a Normally distributed measurement} \tag{3}$$
$$q = \prod_{j,k} p(Y_{obs;j,k}|g); \text{for } j \text{ a Poisson distributed measurement} \tag{4}$$

where, Poisson distributed variables have $\mu = F$ and $\sigma = \sqrt{F}$.

Our strategy is

1 Move calculation of $\sigma$, $s$, $w$, $z$, and $q$, into the second step. The second step is a parallelized calculation and much care needs to be taken to threadproof the calculation once moved into the parallel region. Note: Probability of (measurements $\sim$ Normal) have a component that can be approximated prior to calculating the mean, however, probability of (observation $\sim$ Poisson) requires the true mean, which is not known until step 2 of the calculation (in other documentation step 2 is called the "DO 800 loop", or "support point loop")

2 Define variables required to calculate $q = \prod_{j,k} p(Y_{obs;j,k})|j \sim$ Poisson

3 Define flags required to carry out the current (Normal) calculation only for $j \sim$ Normal, and to calculate the product, $p(Y_{obs}|g) = zq$

4 Define and implement variables required to include calculations on (measurements $\sim$ Poisson) $> P_{thresh}$ in (measurements $\sim$ Normal) calculations

5 Key to this strategy is to also define variables for use in Pmetrics that signal the Fortran code to appropriately update $z$ or $q$

6 Note: All flags and intermediate variables must be threadsafe and communicated across entire progam – from Pmetrics down to, Fortran engine, ODE estimators, statistical programs, and then back up to Pmetrics.

**Progress. January 10, 2019**

Code is put in place to treat all observations as Normal or Poisson measures. Code validation for Normal measures was carried out using the MEROFOS-01 dataset: Nine 240-hour long observations of 2 drugs with a total of 266 measurements, and including an embedded Nelder-Mead optimization to calculate drug interaction. The MEROFOS-01 dataset presented a strong test of code hardness and consistency (i.e. an over-parameterized problem, high level of computational complexity, long calculation time, and many embedded optimizations requiring numerical consistency, ). Results were near identical to previous NPAG versions. The test calculation required the full processing capacity of a MacBook Pro (15-inch, 2017) 3.1 GHz Intel Core i7 with 4 cores (8 threads) for 14.088 hours; significantly faster than the previous program on the same platform, which took 1.9811 days.

```
> PMcompare(54,55,plot=T)
\# Run 54 -- Normal only code; Run 55 -- Poisson ready code
  run type nsub nvar converge.   pval    -2*LL  aic   bic
1  54 NPAG    9   21 TRUE.         NA   1261 1309 1384
2  55 NPAG    9   21 TRUE.       0.56   1261 1309 1384

        popBias popImp popPer_RMSE postBias postImp postPer_RMSE
1      -0.2323  0.269       11.45 -0.07386  0.1741        9.307
2      -0.2322  0.269       11.45 -0.07535  0.1739        9.289
```

Validation on count data requires a trusted simulated dataset. We are in process to find or generate a suitable dataset. In particular, we require a test dataset that will allow us to study the optimization path at the very fine granularity required to compare the base program from the new Poisson-ready software.

Real world code hardening will be carried out using the MEROFOS-01 dataset, which contains both Normal and count data.

Finally, there are two points to mention. First, Poisson-readiness required that we make a change to calculation of the Normally distributed observation standard deviation. In particular, deviation in the base code was calculated using the observation as the mean of the distribution. We now must estimate standard deviation using the prediction as the mean. (The validation above used the observation as a surrogate for mean and the prediction as surrogate for the prediction.) The impact of this change on the search path will need to be studied. The impact of this change on the end stage of the algorithm ought to be minimal. Second, improving processing speed was not our goal here. But the performance improvement clearly indicates we need to seek out other places in the code where large arrays of calculation intermediate values are being copied and passed into the parallelized regions, vs. being calculated inside the parallel region ”on the fly.”

**Progress Notes.**

4/17/2018 Above strategy is coded in idm1*.f and npagranfix*09.f; however all code that can affect computation is commented out in order to check ”base” state of code.

Note. IPAR and RPAR are integer and real valued arrays, respectively, that are passed by the ODE solver between DIFFEQ subroutine and main. Pmetrics creates the subroutine DIFFEQ using user supplied Fortran commands. Any initialization of IPAR or RPAR in the user supplied

Fortran will affect processing inside of the ODE solver, *and* in main, following the first call to DIFFEQ by the ODE solver. NPAG allows for seven output equations ($j$ in the equations above).

(a) Flags to communicate to NPAG from Pmetrics are coded in the IPAR array

  i. IPAR(110 + j).eq.229 indicates the $j^{th}$ output equation represents a Poisson distributed r.v.

  ii. IPAR(120 + j).eq.10 indicate YO(j) is the $\log_{10}$ of the actual measurement

(b) RPAR is used to communicate the values necessary for updating $\sigma$, $z$, and $q$

(c) Note that $Y_{obs} \sim$ Poisson and $Y_{obs} > P_{thresh}$ implies that $Y_{obs} \sim N(F, \sqrt{(F)^2})$, where $P_{thresh}$ is arbitrarily set to 30. Necessary counters and flags are set to integrate these probabilities into $z$

(d) In the Poisson equation, the Gamma function $\Gamma(\%\text{g} + 1.\text{D0})$ is used to approximate the factorial

(e) The special coefficient array (-229, -229, -229, -229) is used in the Pmetrics::model.txt#ERRor block to indicate a Poisson measurement.

(f) In main DO 140, if C0(j).eq.C2(j).eq.C3(j).eq.-229 then 229$\mapsto$IPAR(110+J)

(g) Merofos, Run 53 copies Run 52 to check that code is in "base" state – OUTT* and DEN* files match exactly

4/18/2018 npagran*10.f and idm1*06.f are created from *09 and *05. Poisson code is uncommented for debugging, testing, and validation

(a) SUBROUTINE SUMSQ() written to take in F(:) and put out the sum of the squares. This works because missing values were set to 0 in SUBROUTINE FUNC(), which returned F(:). There is no way to know which increments of F(:) are probabilities and which are normalized deviations. So the calculation of RPAR(128) $= \prod p(y|g)|y \sim$ Poisson and RPAR(129)$= \prod p(y|g)|y \sim$ Normal must be done inside of SUBROUTINE FUNC()

(b) Major bug: *FNTVAL* returned by CALCRF is set to SUMSQ – and I'm not sure why it is not set to $-LL$ or to PYJGX

(c) Fixed by putting back old code that calculated W; and verifying that RPAR(128) and RPAR(129) are calculated correctly

4/19/2018 Development on hold to address Technical Support tickets

4/20/2018 Compile Poisson ready code and duplicate a Normal run

(a) Merofos/Run 54/ should duplicate Run 52, flags for Poisson are checked, but should indicate to use Normal

4/23/2018

(a) Runs 52 (original code), 53 (commented Poisson code), and 54 (commented Poisson, migrated to new machine) match perfectly

(b) Compiling new code (idm1*06.f and npagranfix6*10.f) – fixed a few bugs

(c) Trying to track down a PYJGX=0 bug

(d) Output suggests that all numbers are passed and calculated incorrectly

(e) Poisson calculation uses Ypred instead of Yobs for YMEAN – changed to the observation.

(f) Run 55 – initial cycle output (at very granular level) suggests that sumsq gets calculated correctly – but PYJGX seems to be off after the 14th decimal place – not sure if that is significant enough to change the computation

(g) Run 55 started at about 10:30PM Monday – left running

(h) ILOG to CYCLE 649 is identical ...