FDEGL-User Guide

This file serves as a user guide for the use of fractional non-linear mixed effects models in NONMEM. First the NONMEM control file is described with a typical example provided. Second, the fractional solver subroutine is explained and guidelines are provided for the application of each of the scenarios supported-that is IV bolus with single or multiple doses described by single or multi-compartment models. Finally a short section describing the use of the subroutine in the context of other NONMEM packages such as PsN is also included.

1 The NONMEM Control File

In this section a full working example NONMEM Control File (1) is presented and all the necessary steps for the implementation of the fractional models are thoroughly explained. The subroutine is used by initially specifying to NONMEM that none of it's built-in routines will be used such as $ADVAN$ etc. Instead the command $OTHER = FDEGL.f90$ is written so that NONMEM will use the subroutine named $FDEGL.f90$ located in the same directory as the control file. This last point is necessary before running the estimation process. As with any user-written subroutine, the $PRED$ block must be used. This is were the user is free to write abbreviated code as well as FORTRAN code using the double quotation marks as seen in (1). Here the user must assign the THETA's to fixed effect parameters and the ETA's to model intersubject variability by specifying a random effect model to structural parameters. This is done in an identical manner as in a classic NLME problem studied in NONMEM. In this case however the THETA's must be assigned to a vector -here called VECTRA so as to be compliant with NONMEM's accepted notation- in order to be passed in the subroutine. This means the length of the vector will have to be equal to the number of THETA's. This is shown in the following code snippet

```
MU_1=LOG(THETA(1))
K2=DEXP(MU_1+ETA(1))
MU_2=LOG(THETA(2))
V=DEXP(MU_2+ETA(2))
TVA=THETA(3)
alpha=TVA
```

```
vectra(1)=alpha
vectra(2)=K2
vectra(3)=V
```

It is noted that in this example the MU notation is used since the $SAEM$ method is chosen for the estimation process. However the classic notation can be used as usual for the first order methods FO and $FOCE$.
Following the declaration of all the effects (both fixed and random) an if statement is used in order to set the initial values of variables describing the FDE system. This is done for $t = 0$. These are assigned to a second vector called $VECTRB$ so they also can be passed in the subroutine in vector form. In this example (1) the IV bolus is injected to the central -and only- compartment, hence there is only one element in $VECTRB$. In problems with more dimensions the initial values for the rest of the compartments would also have to be specified. For $t = 0$ the solver is called and returns the initial values of the variables.
For $t > 0$ the user needs to specify three things that the solver accepts as inputs: the numerical step of the solver (tstep), the number of THETA's (Np) i.e the length of $VECTRA$ and finally the number of equations of the FDE system (Dim). In the example of (1):

```
    tstep=0.05
    Np=2
    Dim=1
```

The solver is called by the $CALLFDEGL(\dots)$ command which the user does not alter.
The rest of the blocks are identical to a normal NONMEM control file in which the initial estimates for THETA's, OMEGA's and SIGMA are given by the user and an estimation method is chosen in $\$EST$. At this point **it is important to clarify** that as mentioned in NONMEM's user guide ([1]), in order for any user supplied subroutine (i.e. $\$SUBROUTINE\ OTHER$ is used) which performs calculations outside the control file and/or is not abbreviated code, to work properly, the values of $OPTMAP$ and $ETADER$ must be $\neq 0$. More details concerning $OPTMAP$ and $ETADER$ can be found in (ref NONMEM user guide), but it is vital that the user sets their values to $\neq 0$ as shown in the example provided here

```
    $EST METHOD=SAEM AUTO=1 OPTMAP=2 ETADER=2 INTERACTION
```

In this example an Importance Sampling (IMP) method is also used for a proper calculation of the objective function value after the minimization process, as is often done when using NONMEM's stochastic methods, and is irrelevant to the use of the fractional subroutine. Finally a $\$COV$ section is also used in order for NONMEM to provide the standard errors.

```
$PROBLEM TestFDEGL
$INPUT ID AMT TIME DV EVID MDV
$DATA SIMDATA.TXT
$SUBROUTINES OTHER=FDEGL.f90
$PRED
MU_1=LOG(THETA(1))
K2=DEXP(MU_1+ETA(1))
MU_2=LOG(THETA(2))
V=DEXP(MU_2+ETA(2))
TVA=THETA(3)
alpha=TVA
vectra(1)=alpha
vectra(2)=K2
vectra(3)=V
if(time.gt.0) then
tstep=0.05
Np=3
Dim=1
N=TIME/tstep
" CALL FDEGL(VECTRA,VECTRB,time,ff,int(N),Int(Dim),AMT,int(Np))
else
N=0
VECTRB(1)=AMT
" CALL FDEGL(VECTRA,VECTRB,time,ff,int(N),Int(Dim),AMT,int(Np))
endif
ipre=ff*(V**(-1))
Y=IPRE*(1+EPS(1))
$THETA
       (0.1,9,16)
       (1.5,7,10)
       (0.1,0.5,0.9)
$OMEGA  0.02
  0.02
$SIGMA  0.08

$EST METHOD=SAEM AUTO=1 INTERACTION OPTMAP=1 ETADER=1
$EST METHOD=IMP EONLY=1 NITER=5 ISAMPLE=3000 PRINT=1
$COV
```

Listing 1: Example Control File

Key Steps for the Control File

In this brief example the main alterations of the Control File when using the fractional subroutine, are summarized:

1. The user must declare the $OTHER = FDEGL.f90$ option in the $\$SUBROUTINE$ section.

2. Declare both the fixed and random effects and pass the $THETA's$ in the vector $VECTRA$

3. Choose the time step ($tstep$) of the solver, declare the number of the parameters and the number of the equations (Dim)

4. Pass initial values of variables to a vector $VECTRB$ in the designated section

5. Make sure to set ($OPTMAP \neq 0$ and $ETADER \neq 0$) in the $\$EST$ section

6. Place the subroutine FDEGL.f90 in the same directory as the Control File

2 The fractional subroutine

After configuring the Control File, the user should write the model of fractional differential equations in the FORTRAN subroutine $FDEGL.f90$. This subroutine consists of two parts: The first part is the numerical solver which requires no manipulation by the user, and the second part is the definition of the problem which consists of two FORTRAN functions. These are the function FFUN which is the right hand side $f(t, y)$ of the FDE or system of FDE's, for example

$$ {}_0^C D_t^\alpha y(t) = f(t, y) \tag{1}$$

and the function FFUNJ i.e. the Jackobian of $f(t, y)$. For a system of equations these functions return a vector and matrix respectively whereas for a single equation (like 1) both functions return a scalar. This is done in an identical manner as with any numerical ODE solver in any software (e.g. *ode15s* in MATLAB where the user writes the equations describing the model in a separate function) making the use of the method easy and familiar to the user.

In the FORTRAN file $FDEGL.f90$ there are however 4 functions instead of 2: These are FFUN_1D with FFUNJ_1D and FFUN_ND with FFUNJ_ND. The different type of the return variables (scalar and non-scalar) depending on the dimensions of the problem (single or multi dimensional), is why it necessary to take two cases for the functions FFUN and FFUNJ: One for a single equation model and one for the system of N equations. This is illustrated in the following examples:

Single Equation Model

In the single equation model the user inputs the right-hand side function f(t,y) of (1) in the section labeled *!FUNCTIONS FOR SINGLE-COMPARTMENT!"*. This is shown in (2). The example refers to the model described by the following equation

$$ {}_0^C D_t^\alpha y(t) = -k \cdot y(t), \qquad y(0) = y0 \tag{2}$$

which is the fractional equivalent of a linear elimination equation. The right hand side of (2) is defined in **FFUN_1D** as seen on listing (2) after first declaring the type of each parameter used in the function (The inputs and outputs of the function are declared at the start and no action by the user is required). In Fortran it is especially important for the user to declare the variables used in the code correctly and with the right type, because otherwise errors might arise. The function is computed for each y on line 22 as seen on (2)

```
 1 ! ------------------------------------------------------------------
 2           !FUCNTIONS FOR SINGLE-COMPARTMENT!
 3 ! ------------------------------------------------------------------
 4 function FFUN_1D(t,y,param)
 5 implicit none
 6 INTEGER, PARAMETER  :: dp = SELECTED_REAL_KIND(12, 60)
 7 ! ------------------------------------------------------------------
 8                     !Inputs-No modification by User
 9 ! ------------------------------------------------------------------
10 REAL (dp)  :: ffun_1d
11 REAL (dp)   :: t
12 REAL (dp)   :: y
13 REAL (dp)   :: param(*)
14 ! ------------------------------------------------------------------
15                     !Local vars-Declared by User
16 ! ------------------------------------------------------------------
17
18 REAL (dp)   :: k
19
20 ! -----------------------------------------------------------------
21             ! Definition of Equations--Declared by User
22 ! ------------------------------------------------------------------
23 k = param(2)
24 ffun_1d=-k*y
25 end function FFUN_1D
```

Listing 2: FFUN_1D

In listing (3) the function **FFUNJ _1D** is declared. The Jackobian is in this case simply $\partial f(t,y)/\partial y = -k$. The value is asgined on line 17.

```
 1 function FFUNJ_1D (t,y,param)
 2 implicit none
 3 INTEGER, PARAMETER  :: dp = SELECTED_REAL_KIND(12, 60)
 4 ! ------------------------------------------------------------------
 5                     !Inputs-No modification by User
 6 ! ------------------------------------------------------------------
 7 REAL (dp)  :: ffunj_1d
 8 REAL (dp)   :: t
 9 REAL (dp)   :: y
10 REAL (dp)   :: param(*)
11 ! ------------------------------------------------------------------
12                     !Local vars-Declared by User
13 ! ------------------------------------------------------------------
14
15 REAL (dp)   :: k
16
17 ! ------------------------------------------------------------------
18             ! Definition of Jackobian--Declared by User
19 ! ------------------------------------------------------------------
20
21 k=param(2)
22 ffunj_1d=-k
23 end function FFUNJ_1D
```

Listing 3: FFUNJ_1D

Multi Equation Model

In the case of a model described by a system of FDE's, the user should use the section marked as *!FUNCTIONS FOR MULTI-COMPARTMENT!"*. First for the FFUN_ND function, the user must again declare the variables and set the dimensions of the return variable properly depending on the number of equations. For a system of 2 equations, the variable ffun_ nd will have dimension ffun_nd(2), for 3 dimensions ffun_nd(3) etc. Here, a two dimensional example is presented which is described by the following system of equations

$$\begin{aligned} {}_0^C D_t^\alpha y_1(t) &= -k1 \cdot y_1(t) \\ {}_0^C D_t^\alpha y_2(t) &= k1 \cdot y_1(t) - k2 \cdot y_2(t) \end{aligned} \tag{3}$$

were, $y_1(0) = AMT$ and $y_2(0) = 0$. The listing (4) shows the two steps the user needs to perform: declare the variables in the designated section and write the equations.

```fortran
! ------------------------------------------------------------------
          !FUCNTIONS FOR MULTI-COMPARTMENT!
! ------------------------------------------------------------------

function FFUN_ND(t,y,param)
implicit none
INTEGER, PARAMETER  :: dp = SELECTED_REAL_KIND(12, 60)
! -------------------------------------------------------------------
                    !Inputs-No modification by User
! -------------------------------------------------------------------
REAL (dp)   :: param(*)
REAL (dp)   :: t
REAL (dp)   :: y(*)
! -------------------------------------------------------------------
                !Local Vars-Declared by User
! -------------------------------------------------------------------

REAL (dp)  :: ffun_nd(1)
REAL (dp)   :: k1,k2
! ------------------------------------------------------------------
                !Definition of Equations-Declared by User
! ------------------------------------------------------------------
k1 = param(2)
k2=param(3)

ffun_nd(1)=-k1*y(1)
ffun_nd(2)=k1*y(1)-k2*y(2)

end function FFUN_ND
```

Listing 4: FFUN_ND

Finally the Jackobian is written in FFUNJ_ND. For a two dimensional system of FDE's it will be a 2x2 matrix, for a three dimensional system a 3x3 matrix etc. In this example it's a 2x2 matrix written in (5)

```
function FFUNJ_ND (t,y,param)
implicit none
INTEGER, PARAMETER  :: dp = SELECTED_REAL_KIND(12, 60)
! -----------------------------------------------------------------
                    !Inputs-No modification by User
! -----------------------------------------------------------------
REAL (dp)   :: t
REAL (dp)   :: y(*)
REAL (dp)   :: param(*)

! -----------------------------------------------------------------
                 !Local Vars-Declared by User
! -----------------------------------------------------------------

REAL (dp)   :: k1,k2
REAL (dp)  :: ffunj_nd(2,2)
! -----------------------------------------------------------------
            !Definition of Jackobian-Declared by User
! -----------------------------------------------------------------
k1=param(2)
k2=param(3)

ffunj_nd(1,1)=-k1
ffunj_nd(1,2)=0
ffunj_nd(2,1)=k1
ffunj_nd(2,2)=-k2

end function FFUNJ_ND
```

Listing 5: FFUNJ_ND

Key Steps for the Subroutine

1. Open the subroutine FDE.f90
2. The first part of the code does not require any modification by the user. Go to the second part of the code.
3. Depending on the number of equations go to the designated section.
   - For a single equation:
     !FUNCTIONS FOR SINGLE-COMPARTMENT!"
   - For a system of equations:
     !FUNCTIONS FOR MULTI-COMPARTMENT!"
4. Declare the parameters used in the model and define the function ffun_ 1d or ffun_nd and the Jackobian ffunj_ 1d or ffunj_ nd in the respective FORTRAN functions.

3 Running NONMEM

After the user has configured the Control File and written their model in FDEGL.f90, they can run NONMEM the usual way. If the **execute** command is used to run NONMEM the only modification required is that the user must specify they are using an extra file (FDEGL.f90) with the **extra_file** options as seen in the example below

```
execute ControlFile_Test.txt -extra_files=FDEGL.f90 -retries=5
```

Also extra packages of NONMEM can be easily used in the context of Fractional NLME models simply by declaring the use of the extra subroutine file. For example using the VPC command of PsN

```
vpc ControlFile_Test.txt -extra_files=FDEGL.f90 -samples=500 -rplots=1
```

Any extra options of PsN ([2]) can also be included.

References

[1] Boeckmann A.J. & Bauer R.J. Beal S.L. Sheiner L.B. *NONMEM 7.4 users guides*. URL: https://nonmem.iconplc.com/nonmem743/guides.

[2] *PsN Documentation*. URL: https://uupharmacometrics.github.io/PsN/docs.html.